Modelo de red bayesiana para el análisis de riesgos y resiliencia de la cadena de suministro

Yulayth Katherine Vega Prada

Trabajo de investigación para optar al título de Magister en Ingeniería Industrial

Director

PhD. Henry Lamos Diaz

Grupo de Investigación en:

Optimización y Organización de Sistemas Productivos, Administrativos y Logísticos (OPALO)

Universidad Industrial de Santander

Escuela de Estudios Industriales y Empresariales

Maestría en Ingeniería Industrial

Bucaramanga

2025

Dedicatoria

A mi papá, mi mamá, mi hermana y mi sobrino.

A mis abuelos Luis y Lola, quienes están en el cielo.

Agradecimientos

A mis padres, por su amor incondicional, por ayudarme a alcanzar mis metas y por todos los sacrificios realizados.

A mi hermana por su apoyo constante, y a mi sobrino, quien es una fuente de motivación y felicidad.

A mis abuelos, Luis y Lola, ángeles que desde el cielo me guían y protegen en todo momento.

Al profesor Henry Lamos, por compartir su conocimiento con sus pupilos y por sus valiosas sugerencias y recomendaciones en la realización de esta tesis. Le agradezco por su paciencia, comprensión, disposición y tiempo dedicado, así como por ser un guía fundamental en mi formación académica, tanto de pregrado como de posgrado. Estoy profundamente agradecida.

A mis compañeros de maestría por los momentos compartidos.

A la UIS, al grupo OPALO y a los profesores de la maestría por el apoyo recibido.

A todos, gracias por ser parte de este logro.

Tabla de Contenido

Lista de Tablas

Lista de Figuras

Lista de Siglas y Acrónimos

- **BIP** (Binary Integer Program) – Programa Entero Binario.
- **BN** (Bayesian Network) – Red Bayesiana.
- **CPT** (Conditional Probability Table) – Tabla de Probabilidades Condicionales
- **CI** (Critical Infrastructure) – Infraestructura Crítica.
- **DAG** (Directed Acyclic Graph) – Gráfico Acíclico Dirigido.
- **DBN** (Dynamic Bayesian network) – Red Bayesiana Dinámica
- **ELC** – Empresa de Licores de Cundinamarca.
- **GR** (Global Resilience) – Resiliencia Global.
- **HSC** (Humanitarian Supply Chains) – Cadenas de Suministro Humanitarias.
- **KNIME** (Konstanz Information Miner).
- **LPR** (Loss of Performance during Recovery) – Pérdida de Rendimiento durante la Recuperación.
- **NH** (Natural Hazards) – Peligros Naturales.
- **NPV-LP** (Net Present Value of the Loss of Profit) – Valor Actual Neto de la Pérdida de Ganancias.
- **OEM** (Original Equipment Manufacturer) – Fabricante de Equipo Original
- **OOP** (Object-Oriented Programming) – Programación Orientada a Objetos.
- **OSCI** (Oil Supply Chain Infrastructure) – Infraestructura de la Cadena de Suministro de Petróleo.
- **RASFF** (Rapid Alert System for Food and Feed) – Sistema de Alerta Rápida para Alimentos y Piensos.
- **RIFs** (Risk Influential Factors) – Factores Influyentes en el Riesgo.
- **RSI** (Resilience Index) – Índice de Resiliencia.
- **SC** (Supply Chain) – Cadena de Suministro.
- **SCM** (Supply Chain Management) – Gestión de la Cadena de Suministro.
- **SCRM** (Supply Chain Risk Management) – Gestión de Riesgos de la Cadena de Suministro.
- **SSC** (Software Supply Chain) – Cadena de Suministro de Software.
- **TNT** (Trinitrotoluene) – Trinitrotolueno.
- **TTR** (Time to Recovery) – Tiempo de recuperación.

Glosario

- **Demanda de una unidad a la vez:** Situación en la que los productos se venden o se utilizan uno por uno (Axsäter, 2006).
- **Cálculo do o do-calculus:** Método desarrollado por Judea Pearl para razonar sobre relaciones causales en sistemas complejos.
- **Instancia:** En programación orientada a objetos, se refiere a un objeto creado a partir de una clase (Schildt, 2011).
- **Sistemas de infraestructura crítica (CI, Critical Infrastructure):** Aquellos que están altamente interconectados, ya sea de manera física, geográfica, lógica o a través de una variedad de tecnologías de información y comunicación como la red eléctrica, la red de transporte, la distribución de agua, Internet, entre otros.
- **Inventario de bienes terminados:** Artículos finales listos para venderse (Heizer & Render, 2009).
- **Inventario de materias primas:** Materiales que usualmente se compran, pero aún deben entrar al proceso de manufactura (Heizer & Render, 2009).
- **Inventario de seguridad:** Inventario adicional agregado para satisfacer la demanda (Heizer & Render, 2009)
- **Inventario de trabajo en proceso (WIP, Work in Process Inventory):** Productos o componentes que ya no son materia prima pero no están terminados, les falta parte del proceso de manufactura (Heizer & Render, 2009).
- **Modelo probabilístico:** Modelo estadístico aplicable cuando la demanda del producto o cualquier otra variable se desconoce, pero puede especificarse mediante una distribución de probabilidad (Heizer & Render, 2009).

- **Modelo gráfico probabilístico:** Representación basada en grafos para modelar de manera compacta una distribución compleja (Koller & Friedman, 2009).
- **Nivel de servicio:** Complemento de la probabilidad de un faltante (Heizer & Render, 2009).
- **Fabricante de equipo original (OEM, Original Equipment Manufacturer):** Empresa que manufactura productos que luego son comprados por otra y vendidos al por menor bajo la marca de la empresa compradora.
- **Pgmpy:** Biblioteca de Python que se utiliza para crear y trabajar con modelos gráficos probabilísticos.
- **Posición del inventario:** En el control de inventarios, la situación del stock se caracteriza por la posición del inventario: posición del inventario = stock disponible + pedidos pendientes - pedidos atrasados (Axsäter, 2006).
- **Probabilidad**: Grado de confianza en que un evento incierto ocurrirá.
- **Revisión continua:** Este tipo de revisión implica que el sistema de control de inventario es diseñado de manera que la posición del inventario se monitorea continuamente (Axsäter, 2006).
- **Revisión periódica:** Este tipo de revisión implica que el sistema de control de inventario es diseñado de manera que la posición del inventario se monitorea solo en ciertos momentos dados (Axsäter, 2006).
- **Tiempo de entrega:** Tiempo que transcurre desde que se coloca una orden de reabastecimiento (pedido) hasta la recepción de los bienes (Hillier & Lieberman, 2010).

Resumen

Título: Modelo de red bayesiana para el análisis de riesgos y resiliencia de la cadena de suministro[*]

Autor: Yulayth Katherine Vega Prada[* *]

Palabras Clave: Red bayesiana, Simulación, Cadena de suministro, Análisis de riesgos, Análisis de resiliencia.

Descripción: En este trabajo se desarrolla un modelo de red bayesiana para la predicción de riesgos y el análisis de resiliencia en una cadena de suministro. La metodología comienza con la definición del modelo conceptual, el cual se implementa en el entorno de programación Google Colab, utilizando el lenguaje de programación Python. A través de este entorno se simula la dinámica operativa de una cadena de suministro hipotética, a partir de la cual se obtienen los parámetros de entrada necesarios para construir la red bayesiana. Esta red constituye una representación gráfica y probabilística que modela las relaciones de dependencia entre los actores de la cadena de suministro. Una vez definida, se lleva a cabo el análisis de riesgos y el de resiliencia. En el análisis de riesgos, se determinan las probabilidades marginales de los participantes de la red y se comparan con los resultados obtenidos mediante el método de eliminación de variables de la biblioteca de Python *pgmpy* (diseñada para trabajar con modelos gráficos probabilísticos), evidenciando una concordancia total. Por otro lado, en el análisis de resiliencia, se propone una métrica para su evaluación. Esta investigación ofrece un modelo que combina simulación y redes bayesianas para medir no solo los riesgos de la cadena de suministro, sino también evaluar su capacidad para resistir interrupciones mediante la métrica de resiliencia propuesta.

[*] Trabajo de Grado

[* *] Facultad de Ingenierías Físicomecánicas. Escuela de Estudios Industriales y Empresariales. Maestría en Ingeniería Industrial. Director: Henry Lamos Díaz. PhD. Física-matemática

Abstract

Title: Bayesian network model for the risk analysis and resilience of the supply chain[*]

Author: Yulayth Katherine Vega Prada[* *]

Key Words: Bayesian network, Simulation, Supply chain, Risk analysis, Resilience analysis.

Description: This work develops a Bayesian network model for risk prediction and resilience analysis in a supply chain. The methodology begins with the definition of the conceptual model, which is implemented in the Google Colab programming environment using the Python programming language. Through this environment, the operational dynamics of a hypothetical supply chain are simulated, providing the input parameters necessary to construct the Bayesian network. This network serves as a graphical and probabilistic representation that models the dependency relationships among the supply chain participants. Once defined, both risk analysis and resilience analysis are conducted. In the risk analysis, the marginal probabilities of the network participants are determined and compared with the results obtained using the variable elimination method from the Python library pgmpy (designed for working with probabilistic graphical models), demonstrating complete agreement. On the other hand, in the resilience analysis, a metric is proposed for its evaluation. This research offers a model that combines simulation and Bayesian networks to measure not only the risks in the supply chain but also to assess its ability to withstand disruptions using the proposed resilience metric.

[*] Degree Work

[* *] Faculty of Physicomechanical Engineering. School of Industrial and Business Studies. Master's degree in Industrial Engineering. Director: Henry Lamos Diaz. PhD. Physics-Mathematical.

Introducción

Las cadenas de suministro enfrentan múltiples riesgos que pueden afectar su desempeño (Chopra & Meindl, 2008), siendo particularmente vulnerables debido a la complejidad de las mismas (Hosseini & Ivanov, 2019), derivada del hecho que una cadena está conformada por múltiples actores (eslabones o participantes) interrelacionados.

Las redes bayesianas constituyen un área de investigación en expansión en el análisis de riesgos y la resiliencia de las cadenas de suministro, según se evidencia en la literatura actual. No obstante, a pesar de los avances en este campo, persisten brechas importantes. Por un lado, la mayoría de los estudios aborda los riesgos y la resiliencia de forma separada, y por otro, son escasos los trabajos que consideran estructuras que involucren múltiples actores o que integren la simulación como parte de sus modelos. Hosseini e Ivanov (2020) fueron unos de los primeros autores en analizar la resiliencia de la cadena de suministro basados en redes bayesianas, considerando el efecto dominó que generan las interrupciones en los múltiples eslabones. Sin embargo, su enfoque no incluyó el uso de simulación.

Considerando la complejidad de las cadenas de suministro, la imprevisibilidad de los eventos que desencadenan interrupciones, el impacto que la interrupción de un eslabón puede tener en el desempeño de la cadena, y las brechas existentes en la literatura, se identificó la oportunidad de llevar a cabo el presente proyecto de investigación mediante el cual se desarrolla un modelo basado en redes bayesianas para analizar los riesgos y la resiliencia de manera más integral.

El modelo se desarrolla en el entorno de programación Google Colab utilizando el lenguaje Python. Se utiliza simulación para estimar las probabilidades condicionales de una cadena con múltiples actores, y se emplea una red bayesiana que permite, por una parte, evaluar cómo

diferentes interrupciones afectan la cadena de suministro simulada y, por otra, cuantificar la capacidad de la cadena para resistirlas, medida a través de una métrica de resiliencia.

La investigación contribuye a la literatura en el campo de estudio, representando un aporte valioso a la temática tratada, así como al modelado de redes bayesianas. Además, ha permitido identificar y proponer direcciones para futuras investigaciones que aborden el uso de redes bayesianas en este contexto.

El documento se estructura de la siguiente manera: las secciones 1, 2 y 3 abordan, respectivamente, el planteamiento del problema, los objetivos y la hipótesis de la investigación. La sección 4 presenta un marco teórico con fundamentación conceptual relevante, la sección 5 describe la metodología propuesta para la consecución de los objetivos planteados, la sección 6 incluye la revisión de literatura, la sección 7 expone el modelo conceptual propuesto, la sección 8 detalla el modelo de red bayesiana creado, junto con el análisis de riesgos y el análisis de resiliencia correspondiente. En la sección 9 se lleva a cabo un análisis de sensibilidad, y finalmente, en las secciones 10 y 11 se presentan las conclusiones y recomendaciones derivadas del estudio.

1. Planteamiento del problema

Las cadenas de suministro se exponen a una variedad de riesgos, entre los cuales se incluyen las interrupciones o disrupciones que pueden afectar negativamente su desempeño (Chopra & Meindl, 2008). Estas redes son particularmente vulnerables a dichas situaciones debido a su complejidad (Hosseini & Ivanov, 2019), la cual está relacionada con su estructura y dinámica intrínsecas. Al estar conformadas por múltiples actores (eslabones o participantes) interrelacionados —como proveedores, fabricantes, distribuidores y minoristas—, una interrupción en el flujo de materiales, puede propagar efectos adversos por toda la red (Bugert & Lasch, 2023).

Entre los tipos de disrupciones identificados en la literatura se encuentran los desastres naturales, las guerras, el terrorismo, las fallas en la tecnología de la información, los ataques cibernéticos (Chopra & Meindl, 2008; Bugert & Lasch, 2023), y las crisis de salud pública, como la provocada por la pandemia de COVID-19, que generó interrupciones severas en múltiples cadenas de suministro. En los últimos años, las disrupciones han aumentado en frecuencia, magnitud y gravedad, lo que ha impactado negativamente las operaciones de muchas organizaciones (Hosseini & Ivanov, 2019). Las empresas directamente afectadas sufren pérdidas de producción o problemas de entrega según la gravedad de la crisis (Bugert & Lasch, 2023).

Las redes bayesianas representan un campo de investigación en crecimiento en el análisis de riesgos y en la resiliencia de la cadena de suministro, como lo refleja la literatura reciente; siendo un área de interés importante tanto para académicos como para profesionales. La metodología de redes bayesianas se caracteriza por su capacidad para modelar dependencias en redes complejas (Hosseini & Ivanov, 2020).

A pesar de la creciente atención que han recibido estos temas, la mayoría de estudios en la literatura para medir los riesgos y la resiliencia de la cadena de suministro basados en redes bayesianas son relativamente simples, tratando estos dos aspectos de manera separada, sin incorporar la complejidad de estructuras con múltiples actores que involucre la propagación de interrupciones a lo largo de la red.

Hosseini & Ivanov (2020) han sido los primeros autores que miden la resiliencia de la cadena de suministro considerando la propagación de la interrupción, la cual, según su revisión de literatura previa, se manifiesta como un efecto dominó en la cadena (Hosseini & Ivanov, 2019). Destacan que la resiliencia de la cadena de suministro sigue siendo un área insuficientemente explorada en el ámbito de investigaciones cuantitativas.

Por lo anterior, existe una oportunidad significativa para continuar investigando acerca de los riesgos y la resiliencia de la cadena de suministro desde un enfoque probabilístico basado en redes bayesianas. Actualmente, no se dispone de un estudio que incorpore la propagación de la interrupción en los escalones de la cadena, utilizando redes bayesianas para el análisis de riesgos y resiliencia, y que además desarrolle un modelo de simulación para construir dicha red bayesiana, de manera que las probabilidades condicionales no sean asumidas y estimadas subjetivamente sino mediante los resultados de la simulación de las interacciones entre los actores de la cadena.

Con el fin de abordar la problemática expuesta, en el presente estudio se propone una modificación del trabajo propuesto por (Bugert & Lasch, 2023), desarrollando un modelo de simulación a partir del cual se crea una red bayesiana que permitirá tener una mayor compresión tanto de los riesgos de interrupción que puedan afectar la cadena como de su resiliencia ante estos eventos.

2. Objetivos

2.1 Objetivo General

Desarrollar un modelo de red bayesiana para el análisis de riesgos y resiliencia de la cadena de suministro.

2.2 Objetivos Específicos

- Identificar a través de una revisión de literatura los aportes existentes basados en redes bayesianas relacionados con aplicaciones a riesgos y/o resiliencia de la cadena de suministro.
- Plantear el modelo conceptual estableciendo sus características, definiendo las variables u factores de riesgo relevantes a considerar.
- Desarrollar el modelo de red bayesiana para la predicción de riesgos en una cadena de suministro.
- Validar el modelo de red bayesiana por medio de datos de la literatura y/o información de expertos.

3. Hipótesis

Ante la pregunta de investigación de: ¿Cómo puede desarrollarse un modelo basado en redes bayesianas, construido a partir de simulación, para analizar los riesgos y la resiliencia en cadenas de suministro con múltiples eslabones?

La hipótesis es: El modelo basado en redes bayesianas, construido a partir de simulación, permite mejorar la comprensión de los riesgos y resiliencia en una cadena de suministro.

4. Marco Teórico

4.1 Cadena de Suministro

Una cadena de suministro (SC, Supply Chain) es esa red de flujo físico, información y de efectivo, que va desde los productores para que los artículos y/o servicios lleguen a los consumidores. Puede abarcar varias etapas que incluyen:

- Clientes
- Detallistas/Minoristas
- Mayoristas/Distribuidores
- Fabricantes
- Proveedores de componentes y materias primas

La cadena implica un flujo constante de información, productos y efectivo entre sus diferentes etapas, y está formada por todas aquellas partes involucradas, ya sea de manera directa o indirecta, en la satisfacción de una solicitud de un cliente (Chopra & Meindl, 2008). A continuación, se presenta un ejemplo que proporciona una descripción clara del proceso de una SC, considerando a un cliente que entra a un supermercado colombiano para comprar una caja de Aguardiente Néctar. La SC comienza con el cliente interesado en adquirir el Aguardiente, y continúa con las siguientes etapas:

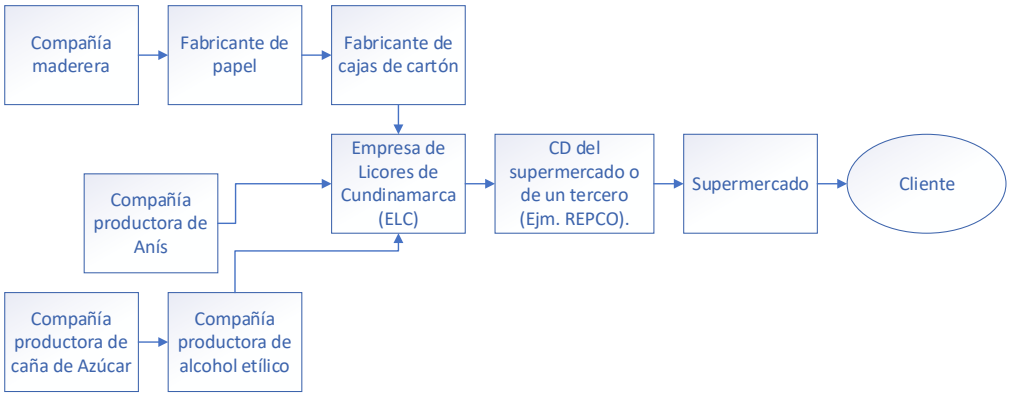
- Supermercado: El cliente visita el punto de venta (minorista), el cual abastece sus estantes con inventarios que pudieron haber sido suministrados desde un almacén de productos terminados de su propiedad o por un distribuidor externo.
- Distribuidor: El distribuidor a su vez es abastecido por el fabricante (La Empresa de Licores de Cundinamarca, ELC) quien es el responsable de la producción del Aguardiente Néctar.

- Fabricante: La ELC recibe insumos clave, como el alcohol y el anís, cada uno suministrado por un proveedor específico. Además, adquiere suministros adicionales como material para el envasado (por ejemplo, cajas de cartón), de otros proveedores.
- Proveedores: Los proveedores del fabricante pueden, a su vez, ser abastecidos por proveedores de niveles más bajos. Por ejemplo, el material para el empaque del Aguardiente proviene de un fabricante de cajas de cartón, quien, a su vez, recibe materia prima de otros proveedores para generarlas, como la compañía fabricante de papel.

Esta estructura de la SC se representa en la Figura 1, donde las flechas indican la dirección del flujo del producto físico.

Figura 1.

Etapas de la cadena de suministro de Aguardiente Néctar



Adaptado de *Administración de la cadena de suministro*, por Chopra, et al., 2008, Pearson E.

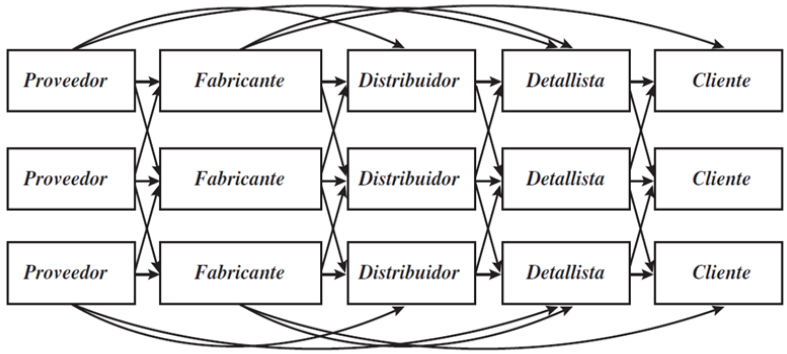
Cada eslabón, actor o participante de la cadena está conectado por el flujo de productos, información y efectivo, los cuales pueden moverse en ambas direcciones, tanto hacia adelante como hacia atrás. El término “cadena de suministro” puede sugerir que un solo participante interviene en cada etapa o que algunas de estas no estén presentes, como en el caso de una cadena que no incluya mayoristas ni detallistas, donde el fabricante surte directamente los pedidos a los clientes. Sin embargo, la mayoría de las cadenas de suministro son, en realidad redes. Por ello podría ser más preciso utilizar el término red de suministro para describir la estructura de la

mayoría de las cadenas, donde, por ejemplo, el fabricante puede recibir material de varios proveedores y luego abastecer a múltiples distribuidores, lo que implica la participación de más de un actor en cada etapa (Chopra & Meindl, 2008).

La Figura 2 ilustra las etapas de una cadena representada como una red de suministro.

Figura 2.

Etapas de la cadena de suministro/red de suministro



Tomado de *Administración de la cadena de suministro*, por Chopra, et al., 2008, Pearson E.

4.1.1 Gestión de la cadena de suministro

El manejo de la SC es un asunto importante en la actualidad, ya que muchas empresas logran una ventaja competitiva significativa mediante la forma en que gestionan las operaciones de su cadena. En este sentido, la gestión de la cadena de suministro (SCM, Supply Chain Management) significa darle manejo al flujo de información, materiales y servicios desde los proveedores de materia prima, pasando por fábricas y bodegas, hasta el usuario final (Jacobs & Chase, 2000).

Combinando diferentes definiciones desarrolladas por varios autores, Tang (2006) define la SCM como “la gestión de materiales, información y flujos financieros a través de una red de organizaciones (es decir, proveedores, fabricantes, proveedores de logística, mayoristas/distribuidores, minoristas) que tiene como objetivo producir y entregar productos o servicios para los consumidores. Incluye la coordinación y colaboración de procesos y actividades

a través de diferentes funciones como marketing, ventas, producción, diseño de productos, adquisiciones, logística, finanzas y tecnología de la información dentro de la red de organizaciones".

Como todo proceso, la SC debe ser gestionada de manera efectiva. El objetivo de la SCM se ha expandido más allá del ahorro en costos a corto plazo, a beneficios estratégicos a largo plazo, como lograr un alto nivel de resiliencia ante la incertidumbre y la dinámica que enfrenta la SC, lo que a su vez conduce a mejoras en la entrega del producto final o el servicio al cliente (Hosseini & Ivanov, 2019).

Por otra parte, en las cadenas de suministro, el proveedor desempeña un rol fundamental. Por ejemplo, si se planea producir una cierta cantidad de unidades para determinada fecha, es necesario que los insumos requeridos para producir dichas unidades se entreguen a tiempo. Esto requiere una buena integración entre proveedores y clientes, así como una gestión adecuada de los riesgos, de tal forma que se disponga del material en el momento preciso y así poder cumplir con los requerimientos (Jacobs & Chase, 2000).

4.1.2 Aprovisionamiento

"El aprovisionamiento es un conjunto de procesos que se requieren para comprar bienes y servicios". Las decisiones de aprovisionamiento tienen un impacto en varios aspectos como, el costo de los productos vendidos, la calidad de los productos, los inventarios, entre otros. Una de las decisiones clave de aprovisionamiento es la selección de proveedores. Un proveedor deficiente perjudica la organización, afectando su capacidad de respuesta e incrementando, por ejemplo, la cantidad de inventario que la SC debe mantener. Es necesario, entonces, decidir el número de proveedores que se tendrán para una actividad o material particular, así como definir los criterios con los cuales se evaluarán y seleccionarán dichos proveedores (Chopra & Meindl, 2008).

4.1.3 Riesgos de la cadena de suministro

En los últimos años, algunas cadenas de suministro han crecido, convirtiéndose en redes globales cada vez más complejas (Badurdeen et al., 2014). Las múltiples colaboraciones y relaciones entre los participantes hacen que los eventos disruptivos en un eslabón impacten o se propaguen hacia otros eslabones. Asimismo, la imprevisibilidad de estos eventos contribuye a que el desempeño de las cadenas sea cada vez más incierto.

El aumento en la frecuencia y las graves consecuencias de las interrupciones ha generado un creciente interés en estudiar el riesgo asociado a la SC (Heckmann, Comes & Nickel, 2015), considerándose un tema cada vez más relevante. Para frenar las consecuencias adversas de la propagación de las interrupciones o los efectos de las interrupciones en la SC y mantener la competitividad de las empresas, la estimación y análisis del riesgo de interrupción ha tenido una alta prioridad en la SCM (Liu et al. 2020).

Heckmann, Comes & Nickel (2015) revisaron estudios que clasifican y definen el riesgo de la SC, así como enfoques para su cuantificación. Encontraron que mientras algunos autores analizan las consecuencias de un evento sobre una sola empresa, otros se centran en el desempeño de la cadena en su conjunto, considerando los efectos que se propagan a través de la red de suministro. Concluyen que, no se tiene una definición unánime de riesgo en la SC, lo que hace que este siga siendo un aspecto difícil de evaluar, monitorear, controlar y, aún más, de representar en modelos matemáticos de decisión.

A continuación, en la Tabla 1, se presentan algunas definiciones de riesgo de la SC disponibles en la literatura, destacando a March & Shapira (1987) como unos de los primeros autores en abordar este tema.

Tabla 1.

Definición de riesgo de la cadena de suministro

Autores	**Definición de riesgo de la cadena de suministro**
March & Shapira (1987)	Variación en la distribución de los posibles resultados de la cadena de suministro, su probabilidad y sus valores subjetivos.
Jüttner, Peck & Christopher (2003)	La posibilidad y el efecto de un desajuste entre la oferta y la demanda.
Wagner & Bode (2008)	La exposición a eventos no deseados que pueden resultar en interrupciones o en un deterioro del rendimiento.
Manuj & Mentzer (2008)	Cualquier fuente de incertidumbre que pueda afectar los resultados de la cadena, desde variabilidad en la demanda hasta fallos en el suministro o fluctuaciones en los precios.
Heckmann, Comes & Nickel (2015)	La probabilidad de ocurrencia de un evento disruptivo y su impacto (gravedad de las consecuencias).
Hosseini & Ivanov (2019)	Probabilidad y severidad de eventos disruptivos.

Por otro lado, Lockamy & McCormack (2012) plantean que los riesgos de la SC se derivan de fuentes de incertidumbre tanto internas como externas. Las fuentes internas pueden incluir cambios en la disponibilidad de capacidad, interrupciones en los flujos de información y reducciones de la eficiencia operativa, mientras que las fuentes externas pueden ser causadas por acciones de los competidores, fluctuaciones de precios, cambios en el entorno político y variaciones en la calidad de los proveedores, entre otros.

Gestión de riesgos de la cadena de suministro

Los riesgos de la SC pueden abordarse desde dos perspectivas, los riesgos operativos que están relacionados con la incertidumbre inherente de la SC, tal como demanda incierta, suministro incierto y costo incierto; y los riesgos de interrupción, asociados con interrupciones causadas por crisis económicas o desastres naturales como terremotos, inundaciones, huracanes, ataques terroristas etc. El impacto en el negocio, asociado a los riesgos de interrupción, en la mayoría de los casos, es mucho mayor que el de los riesgos operativos. En este sentido, Tang (2006) define la gestión de riesgos de la cadena de suministro (SCRM, Supply Chain Risk Management) como "la

gestión de los riesgos de la SC a través de la coordinación o colaboración entre los socios de la cadena para garantizar la rentabilidad y la continuidad".

Para mitigar el impacto de las interrupciones de la SC frente a diversos tipos de riesgos como ciclos económicos inciertos, fluctuaciones en la demanda, desastres naturales etc, se han desarrollado estrategias y modelos de gestión de riesgos. Tang (2006) propone cuatro enfoques: gestión de la oferta, coordinando con socios para asegurar un suministro eficiente de materiales; gestión de la demanda, trabajando con socios intermedios para influir en la demanda; gestión de productos, modificando el diseño del producto o del proceso para que sea más fácil hacer que la oferta satisfaga la demanda; y gestión de la información para mejorar la coordinación a lo largo de la SC.

Se ha llevado a cabo un trabajo considerable en la SCRM proporcionado modelos cualitativos y cuantitativos para la identificación, priorización y mitigación de los riesgos de la SC. Hosseini & Ivanov (2020) exponen ventajas metodológicas del uso de BNs para gestionar la incertidumbre asociada con una SC compleja sobre otros métodos existentes como el modelado de regresión, cadenas de Markov, simulaciones Monte Carlo y modelado de ecuaciones estructurales.

Por otra parte, de acuerdo con lo que plantea Sharma, Routroy & Chanda (2022), estudios como el modelo estructural interpretativo (Babu, Bhardwaj & Agrawal, 2020), modelos de jerarquía analítica (Dong & Cooper, 2016), y modelos difusos (Heidari, Khanbabaei & Sabzehparvar, 2018) han contribuido a desarrollar una mejor comprensión de la propagación del riesgo de la SC.

4.1.4 Resiliencia de la cadena de suministro

La resiliencia se relaciona con la capacidad de un sistema (material, red, individuo, empresa) para ajustar o mantener funciones esenciales en condiciones estresantes y duras. Es un

concepto que se aplica en diferentes disciplinas, como ecología, ingeniería, sociología, psicología y economía (Heckmann, Comes & Nickel, 2015). No existe una definición única y concreta para el concepto de resiliencia de la SC (Hosseini & Ivanov, 2019). En el ámbito de estudio, el término ha sido definido de diferentes maneras; en la Tabla 2 se presenta un resumen con algunas de ellas.

Tabla 2.

Definición de resiliencia de la cadena de suministro

Autores	**Definición de resiliencia de la cadena de suministro**
Brandon-Jones. et al (2014)	Capacidad de un sistema para volver a su estado original, dentro de un período de tiempo aceptable después de ser perturbado.
Heckmann, Comes & Nickel (2015)	Capacidad de la red para superar una vulnerabilidad; donde vulnerabilidad es una palabra utilizada para describir el grado en que la cadena es susceptible a un evento de riesgo especifico.
Hosseini & Ivanov (2019).	Capacidad de la SC para resistir, adaptarse y recuperarse de interrupciones a un costo mínimo, garantizando que se satisfaga la demanda del cliente.

La resiliencia no se ha explorado completamente en la investigación cuantitativa de la SCRM hasta el momento. En particular, las características dinámicas de los modelos de SC bajo disrupción y sus procesos de recuperación han sido poco estudiados (Behzadi, O'Sullivan & Olsen, 2020). A pesar de la escasez general de este tipo de trabajo, existen ciertas métricas en la literatura.

Behzadi, O'Sullivan & Olsen (2020) proponen una métrica de resiliencia de la SC llamada valor actual neto de la pérdida de ganancias (NPV-LP), evaluada en pequeño problema que consta de un nodo de suministro y uno de demanda de un bien perecedero, en un horizonte de varios períodos.

Fang & Zio (2019) cuantifican la resiliencia de una infraestructura critica (CI, Critical Infrastructure) frente a peligros naturales (NH, Natural Hazards) basándose en su proceso de desempeño durante un periodo de tiempo [0, T]. Estos NH se clasifican en: geofísicos (terremotos, volcanes y tsunamis), meteorológicos (tormentas tropicales, tornados, ventiscas, tormentas de hielo y sequías), hidrológicos (inundaciones), biológicos (epidemias y plagas de insectos) y

extraterrestres (meteoritos). Miden la pérdida de rendimiento durante la recuperación (LPR, Loss of Performance during Recovery) en términos de la pérdida de ganancias desde el inicio de la interrupción hasta el final del período de recuperación.

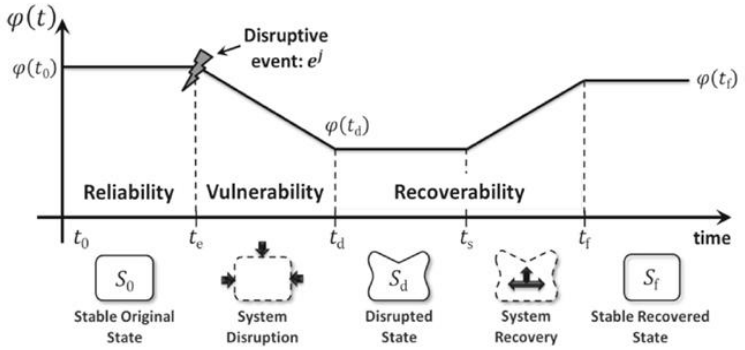
Nan & Sansavini (2017) proponen un método cuantitativo para la evaluación de la resiliencia de un sistema, utilizando una métrica integrada adimensional que combina varias medidas que contribuyen por igual a la resiliencia.

Ojha et al. (2018) miden la resiliencia de los nodos de la SC utilizando un Índice de Resiliencia (RSI, por sus siglas en inglés). El RSI se calcula teniendo en cuenta el nivel de servicio del nodo.

Henry & Ramirez-Marquez (2012) introdujeron una métrica que cuantifica la resiliencia de un sistema. Definieron el desempeño del sistema en el tiempo t como $\varphi(t)$ y establecieron tres estados de transición: (1) estado estacionario, en el que el sistema opera normalmente (antes de la disrupción); (2) estado de vulnerabilidad, en el que sistema se ve afectado por un evento disruptivo y finalmente, (3) estado de recuperabilidad, donde la función de servicio del sistema aumenta desde $\varphi(t_d)$ hasta $\varphi(t_f)$ (ver Figura 3) (Hosseini & Ivanov, 2019).

Figura 3.

Rendimiento y transiciones de estado para describir la resiliencia de un sistema



Adaptado de Henry & Ramirez-Marquez (2012), en *A new resilience measure for supply networks with the ripple effect considerations: a Bayesian network approach*, por S. Hosseini y D. Ivanov, 2019, Annals of Operations Research.

La Tabla 3 presenta un resumen de las métricas existentes en la literatura, asociadas con la resiliencia de la SC, revisadas previamente.

Tabla 3.

Métricas de resiliencia

Autores	Métrica de resiliencia	Descripción
Behzadi, O'Sullivan & Olsen (2020)	$NPV - LP$ (Valor actual neto de minimización de pérdida de ganancias)	Medida que permite calcular el impacto financiero de las pérdidas de beneficio en el tiempo, considerando el valor temporal del dinero.
Fang y Zio (2019)	R^{k}	La resiliencia R^{k} de una Infraestructura Crítica k durante [0, T] frente a un peligro natural se cuantifica como el cociente entre dos curvas de desempeño: $Ptarget(t)\ o$ curva sin interrupciones y $Preal(t)$ que describe el cambio en el rendimiento bajo interrupciones y procesos de restauración. Donde $\mathbb{T}$ es el conjunto de todos los periodos de tiempo discretos dentro del horizonte de desastre, definido como el momento en que todos los componentes fallidos se han restaurado completamente. $$R^{\mathrm{k}} = \frac{\sum_{t\in\mathbb{T}} P_{real}(t)}{\sum_{t\in\mathbb{T}} P_{target}(t)}$$
Nan & Sansavini (2017)	GR (Resiliencia Global)	El índice de resiliencia (GR, Global Resilience) combina varias medidas que evalúan la resistencia, adaptación y recuperación del sistema frente a eventos disruptivos, considerando la robustez R, la velocidad de recuperación $RAPI_{RP}$, la capacidad de recuperación RA, la velocidad de pérdida $RAPI_{DP}$ y la pérdida de rendimiento $TALP$. $$GR = R \times \frac{RAPI_{RP}}{RAPI_{DP}} \times \frac{1}{TALP} \times RA$$
Ojha et al. (2018)	RSI_k (Índice de resiliencia del nodo k)	El RSI_k es un valor normalizado entre 0 y 1 que mide la capacidad de un nodo k para mantener su nivel de servicio después de haber sido afectado por una disrupción en un período de tiempo. Donde w_0 es la semana en que ocurre la disrupción en el nodo k, y w_n la semana en la que finaliza, y el nodo se ha recuperado completamente. SL_{k0} corresponde al nivel de servicio del nodo k en condiciones normales (sin disrupción), y SL_{kw} al nivel de servicio del nodo k en una semana específica w durante la disrupción. $$RSI_k = 1 - \frac{\int_{w_0}^{w_n}(SL_{k0} - SL_{kw})dw}{SL_{k0}(w_n - w_0)}$$
Henry & Ramirez-Marquez (2012)	$Я(t_r\|e^j)$	La resiliencia $Я(t_r\|e^j)$ mide la relación entre la recuperación y la pérdida en términos de la función de servicio. Representa la proporción de la función de servicio que se ha recuperado de su estado disruptivo. El valor de $Я(t_r\|e^j)$ correspondiente a una determinada función de servicio evaluada en t_r, donde $t_r\,\epsilon(t_d, t_f)$, bajo el evento disruptivo e^j se calcula como: $$Я(t_r\|e^j) = \frac{\varphi(t_r\|e^j) - \varphi(t_d\|e^j)}{\varphi(t_0) - \varphi(t_d\|e^j)}$$

Adaptado de Behzadi, O'Sullivan & Olsen (2020)

Como se observa en la Tabla 3, las métricas en la literatura utilizadas para evaluar la resiliencia de la SC abarcan una variedad de enfoques. Considerando la resiliencia como la capacidad de recuperarse rápida y efectivamente de la interrupción, el tiempo de recuperación (TTR, time to recovery) también es adecuado como una métrica de resiliencia, ya que cuantifica el tiempo requerido para que una red de suministro vuelva a funcionar normalmente después de una interrupción. (Behzadi, O'Sullivan & Olsen, 2020).

Por otro lado, son varias las estrategias empleadas por las redes de suministro para gestionar los riesgos asociados con interrupciones significativas y lograr una mayor resiliencia. Estas estrategias se pueden clasificar en dos grupos. Las estrategias de mitigación o acciones preventivas que se implementan antes de que ocurra una interrupción, independientemente de si se materializa o no. En contraste a las estrategias de contingencia, que se activan cuando ocurre la interrupción, y el propósito es restaurar la SC a su estado inicial (Fattahi, Govindan, & Keyvanshokooh, 2020).

Losada, Scaparra & O'Hanley (2012) abordan el problema de reducir el impacto de las fallas de componentes en los sistemas de servicio y suministro, decidiendo qué componentes reforzar o proteger para acelerar la recuperación de un sistema. Para ello presentan un modelo de programación lineal binivel entero mixto que integra estrategias de mitigación y contingencia para enfrentar múltiples interrupciones a lo largo del tiempo.

La resiliencia de la cadena de suministro es un área de estudio que puede generar beneficios significativos para las empresas. Es necesario llevar a cabo investigaciones más completas para aprovechar el potencial de este interesante ámbito (Ribeiro & Barbosa-Povoa, 2018).

4.2 Principios Subyacentes de Redes Bayesianas

A continuación, se presentan fundamentos teóricos importantes que sustentan la aplicación y el uso de redes bayesianas.

4.2.1 Conceptos básicos de la teoría de la probabilidad

La teoría de la probabilidad surge de situaciones de la vida real, en las que se realiza un experimento y el resultado observado es incierto. Al estudiar experimentos, se está interesado en un subconjunto particular de resultados, llamados eventos. Se define E como un evento o cualquier subconjunto del espacio muestral S (todos los resultados individuales posibles del experimento), al cual es posible asignar probabilidades que satisfacen las siguientes propiedades (Montgomery & Runger, 1996):

1. $P(\mathrm{S}) = 1$

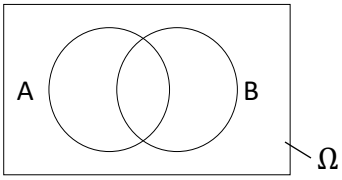
2. $0 \leq P(E) \leq 1$

3. Para dos eventos E_1 y E_2 con $E_1 \cap E_2 = \emptyset,\ P(E_1 \cup E_2) = P(E_1) + P(E_2)$

4.2.2 Probabilidad condicional

La probabilidad de que ocurra el evento B, dado que se sabe que el evento A ha ocurrido, se calcula como la proporción de la probabilidad de que ambos eventos ocurran respecto a la probabilidad de que A ocurra (Koller & Friedman, 2009). Formalmente, la probabilidad condicional de **B** dado **A** se define como:

$$P(B|A) = \frac{P(A \cap B)}{P(A)}$$



$A \cap B \neq 0$

De la definición se tiene que:

$$P(A \cap B) = P(A)P(B|A)$$

Esta igualdad se conoce como la regla de la cadena de probabilidades condicionales.

Específicamente, si $A_1, \ldots, A_k$ son eventos, se puede escribir:

$$P(A_1 \cap \ldots \cap A_k) = P(A_1)P(A_2|A_1) \ldots P(A_k|A_1 \cap \ldots \cap A_{k-1})$$

Otra deducción de la definición de probabilidad condicional es la regla de Bayes (Ecuación 1), también conocida como teorema de Bayes o ley de Bayes. Su nombre se da en honor a Thomas Bayes (1702-1761) quien fue el autor de su formulación (Russell & Norvig, 2003):

$$P(B|A) = \frac{P(A|B)P(B)}{P(A)} \quad (\mathbf{1})$$

4.2.3 Modelos gráficos probabilísticos

Los modelos gráficos probabilísticos utilizan una representación basada en grafos para modelar de manera compacta una distribución compleja. Aprovechan el hecho de que, en muchas distribuciones, las variables interactúan directamente solo con algunas de las demás. Esto permite que dichas distribuciones se representen de manera compacta y clara usando un enfoque basado en grafos (Koller & Friedman, 2009).

El marco de los modelos gráficos probabilísticos es amplio y abarca diversos tipos de modelos y métodos. Una de estas representaciones gráficas de distribuciones son las redes bayesianas (Koller & Friedman, 2009).

4.3 Redes Bayesianas

Una red bayesiana (BN, Bayesian Network) es un grafo acíclico dirigido (DAG, Directed Acyclic Graph) que muestra relaciones probabilísticas entre nodos o variables de interés en un problema de razonamiento incierto (Pai et al. 2003). De acuerdo con Russell & Norvig (2003), una BN se compone de:

1. Un conjunto de variables aleatorias o nodos $\{x_1, x_2, \dots x_n\}$
2. Un conjunto de arcos, enlaces dirigidos o flechas que conectan pares de nodos. Un arco saliente de x_i a x_j indica la dependencia entre estas dos variables, de manera tal que x_i es el padre de x_j y por tanto x_j es el hijo de x_i.
3. Distribuciones de probabilidad condicionales de cada nodo.

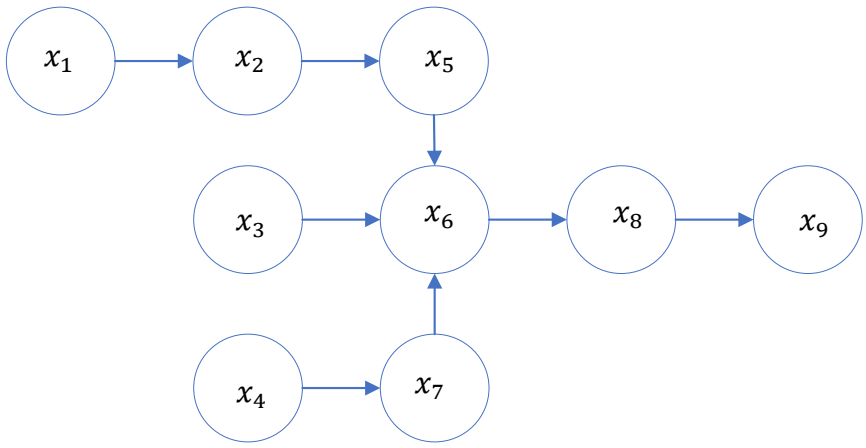
En términos generales hay tres clases de nodos en una BN:

- Nodos sin hijos (nodos hoja)
- Nodos sin nodos padres (nodos raíz)
- Nodos con nodos padre e hijo (nodos intermedios) (Russell & Norvig, 2003).

En la Figura 4 se puede observar una BN del ejemplo de cadena de suministro de la ELC. La red contiene nueve nodos: x_1, x_3 y x_4 son nodos raíz, x_2, x_5, x_6 , x_7 y x_8 son nodos intermedios, y x_9 es un nodo hoja. Cada nodo representa un actor de la SC. En particular, la variable x_1 corresponde a la compañía maderera, x_2 a la fábrica de papel, x_5 al fabricante de cajas de cartón, x_3 a la compañía productora de anís, x_6 a la ELC, x_4 y x_7 a las compañías productoras de caña de azúcar y alcohol etílico, respectivamente, x_8 al centro de distribución, y, finalmente, x_9 representa el supermercado donde el cliente compra el Aguardiente.

Figura 4.

Ejemplo de BN



Adaptado de Hosseini & Ivanov (2019), en *A new resilience measure for supply networks with the ripple effect considerations: a Bayesian network approach*, por S. Hosseini & D. Ivanov, 2019, Annals of Operations Research.

Las redes bayesianas (BNs, Bayesian Networks) se fundamentan en el teorema de Bayes, utilizado para calcular probabilidades condicionales en función de un conjunto de evidencias. Estas han evolucionado como una herramienta eficaz para analizar la incertidumbre (Pai et al. 2003), siendo reconocidas como una metodología rigurosa para la cuantificación de riesgos, el modelado de incertidumbre, y ayudar al proceso de toma de decisiones (Hosseini & Ivanov, 2020). Las BNs son útiles para el análisis de riesgos de sistemas complejos, debido a que pueden modelar fácilmente las relaciones de dependencias entre los componentes del sistema, siendo capaces de describir las causas y los efectos de la salida del mismo, proporcionando cuantificaciones rigurosas de los riesgos (Hosseini & Ivanov, 2019).

A pesar de que las BNs son un modelo común para el análisis de riesgos, no se ha profundizado lo suficiente en su adaptación al riesgo de suministro (Sharma, Routroy & Chanda, 2022). Adicionalmente, la metodología de BN permite evaluar la resiliencia con la propagación de la interrupción (Hosseini & Ivanov, 2019), pero aún existe una considerable oportunidad para investigar más en la literatura sobre este aspecto.

4.3.1 Distribución de probabilidad conjunta completa

La distribución de probabilidad conjunta completa especifica la probabilidad de cada asignación de valores a todas las variables aleatorias, puede usarse para responder cualquier pregunta sobre el dominio. La red bayesiana proporciona una descripción completa del mismo, y es una representación de la distribución conjunta, por lo cual puede también ser utilizada para responder cualquier pregunta, a partir de la sumatoria de todas las entradas conjuntas relevantes (Russell & Norvig, 2003). Sea X una variable aleatoria, la distribución conjunta de una Red Bayesiana está dada por:

$$P(X_1, X_2, \ldots, X_n) = \prod_{i=1}^{n} P(X_i | padres(X_i)),$$

Cada entrada de la distribución de probabilidad conjunta puede calcularse a partir de la información de la red. Una entrada genérica en la distribución conjunta es la probabilidad de una conjunción de asignaciones concretas a cada variable, tal como $P(X_1 = x_1 \wedge \ \ldots \ \wedge X_1 = x_n)$. El valor de la entrada está dado por (Russell & Norvig, 2003):

$$P(X_1 = x_1 \wedge \ \ldots \ \wedge X_n = x_n) = \prod_{i=1}^{n} P(X_i = x_i | padres(X_i)),$$

4.3.2 Inferencia en redes bayesianas

La inferencia probabilística tiene como objetivo calcular la distribución de probabilidad a posteriori para un conjunto de variables pregunta, dado un evento observado (es decir, alguna asignación de valores para un conjunto de variables de evidencia). En este contexto, se puede denotar a X como la variable pregunta, a **E** como el conjunto de variables de evidencias $E_1, \ldots, E_m$, a $\boldsymbol{e}$ como el evento observado, y a **Y** como un conjunto de variables no evidencia $Y_1 \ldots, Y_l$ (también conocidas como variables ocultas); siendo el conjunto completo de variables en la red el siguiente: (Russell & Norvig, 2003):

$$\mathbf{X} = \{X\} \cup \mathbf{E} \cup \mathbf{Y}$$

4.3.3 Algoritmo de eliminación de variables

El algoritmo de eliminación de variables es un método utilizado para responder a preguntas en redes bayesianas. Una pregunta típica consiste en determinar la distribución de probabilidad a posteriori $\boldsymbol{P}(X \mid \boldsymbol{e})$. Mediante este algoritmo, es posible calcular la distribución de probabilidad de una variable pregunta X, dada una evidencia e, en una red bayesiana rb (Russell & Norvig, 2003).

Su nombre de eliminación de variables proviene del hecho que, durante el cálculo de la probabilidad, se eliminan aquellas variables que no influyen en el resultado final, simplificando así el proceso computacional.

En el siguiente pseudocódigo se describe el funcionamiento del algoritmo de eliminación de variables (Russell & Norvig, 2003):

```
función PREGUNTAR-POR-ELIMINACIÓN(X, e, rb) devuelve una distribución
sobre X
  entradas: X, la variable pregunta
            e, evidencia establecida como un evento
            rb, una red bayesiana que especifique una distribución
conjunta
            P(X1..., Xn)
  factores ← [ ]; vars ← INVERTIR(VARS[rb])
  para cada var en vars hacer
    factores ← [CREAR-FACTOR(var, e)_ factores]
    si var es una variable oculta entonces factores ← SUMA(var,
factores)
  devolver NORMALIZAR(PRODUCTO-PUNTO-A-PUNTO (factores))
```

5. Metodología

El desarrollo de esta investigación se basa en una metodología de cinco etapas. La primera etapa corresponde al primer objetivo, que comprende el proceso de revisión de literatura. Su propósito es identificar los estudios científicos disponibles que emplean redes bayesianas para el análisis de riesgos y/o resiliencia de la cadena de suministro, contextualizando el estado actual de la temática investigada. Las etapas dos, tres y cuatro están diseñadas para responder, respectivamente, a los objetivos segundo, tercero y cuatro de la investigación. Finalmente, la quinta etapa se centra en documentar cada una de las etapas previas, formular las conclusiones y recomendaciones del estudio, y elaborar un artículo que sintetiza el proyecto.

Etapa uno: Revisión de literatura

Para recopilar información científica relevante para la investigación, que proporcione un contexto actualizado sobre el tópico investigado (redes bayesianas, análisis de riesgos y resiliencia en la cadena de suministro), se construyó una ecuación de búsqueda que se implementó en las bases de datos Scopus y Web of Science con la correspondiente recuperación de los artículos arrojados y su posterior lectura y selección. Los pasos seguidos en esta etapa fueron los siguientes:

- Identificación de palabras clave
- Construcción de la ecuación de búsqueda e implementación en Scopus y en Web Of Science
- Recuperación de los artículos obtenidos a partir de las búsquedas realizadas.
- Lectura de palabras clave y resúmenes (abstracts) para la selección de artículos de artículos relevantes para la revisión literaria.

Adicionalmente, en esta etapa se utilizó la metodología de "bola de nieve" para identificar y acceder a documentos adicionales relevantes a partir de las citas bibliográficas de los artículos de las bases de datos.

Etapa dos: Elaboración de modelo conceptual

En esta etapa se define y describe el diagrama del modelo conceptual que guía el proceso metodológico basado en redes bayesianas para el análisis de riesgos y resiliencia en una cadena de suministro. El modelo conceptual involucra:

- La delimitación de los supuestos de la dinámica de funcionamiento de la cadena de suministro
- La definición de los parámetros de entrada del modelo.
- La creación de clases/etapas en Python, métodos de cada clase y creación de instancias.
- La definición de variables a involucrar para la creación de la red bayesiana, así como los estados asociados a cada una de ellas.

Etapa tres: Formulación del modelo

Esta etapa consiste en construir en la red bayesiana, la formulación del respectivo análisis de riesgos y análisis de resiliencia con el desarrollo asociado. Incluye:

- Obtención de probabilidades marginales y condicionales de los actores de la cadena.
- Creación de la Red Bayesiana en Python
- Formulación de análisis de riesgos en Python
- Formulación de análisis de resiliencia Python

Etapa cuatro: Validación del modelo

En esta etapa se realiza un análisis de sensibilidad mediante el cual se valida el modelo.

Etapa cinco: Documentación

Esta etapa corresponde al registro en el documento del proyecto del modelo conceptual propuesto, la red bayesiana generada, los resultados obtenidos, así como su validación. Adicionalmente, en esta última fase se elabora un artículo de investigación donde se resume el proyecto.

6. Revisión de Literatura

Para alcanzar el primer objetivo de esta investigación, se realizó una revisión de la literatura con el fin de identificar y contextualizar los aportes existentes relacionados con la modelización mediante redes bayesianas en el contexto de riesgos y/o resiliencia de la cadena de suministro. La revisión incluyó una búsqueda sistemática en las bases de datos Scopus y Web of Science, utilizando una ecuación de búsqueda diseñada para este propósito. Las palabras clave empleadas fueron:

- *Supply chain management OR supply chain*
- *Resilience OR risk*
- *Bayes* OR bayes model OR bayes* network**

Estas palabras clave se seleccionaron por su pertinencia al tema de investigación. Se combinaron de manera lógica para construir la ecuación, utilizando comillas para buscar frases exactas, paréntesis para agrupar términos, operadores booleanos para estructurar la búsqueda, y truncadores (*) para ampliarla, permitiendo contemplar diferentes variaciones posibles de los términos.

La ecuación se restringió a los campos de título, resumen y palabras clave, con el objetivo de asegurar que los resultados fueran relevantes para la investigación, y evitar aquellos no relacionados con el tema central. Además, la ventana de búsqueda se limitó a publicaciones en inglés comprendidas entre 2004 y diciembre de 2023.

La estructura final de la ecuación, adaptada para cada herramienta (Scopus o Web of Science), se presenta a continuación:

Scopus: TITLE-ABS-KEY (("supply chain management" OR "supply chain") AND (resilience OR risk) AND (bayes* OR "bayes model" OR "bayes* network* ")) AND PUBYEAR > 2003 AND PUBYEAR < 2024 AND (LIMIT-TO (LANGUAGE, "English"))

Web of Science: TS=(("supply chain management" OR "supply chain") AND (resilience OR risk) AND (bayes* OR "bayes model" OR "bayes* network*")) AND LA= (English)

Como resultado de la implementación de la ecuación en las bases de datos mencionadas, se obtuvieron 266 registros en Scopus y 178 en Web of Science, para un total de 444 documentos. Tras eliminar 116 duplicados, se identificaron 328 documentos únicos (el análisis bibliométrico correspondiente se puede consultar en el Apéndice A). Luego de la lectura de palabras clave y resúmenes, de los 328 documentos, se seleccionaron los considerados pertinentes para su análisis en profundidad; y utilizando el método de búsqueda de información conocido como bola de nieve que permite encontrar nueva bibliografía relevante a partir de las fuentes iniciales, se accedió a otros documentos relevantes.

En la revisión se encuentra que la aplicación de redes bayesianas en la gestión de la cadena de suministro ha demostrado ser un enfoque prometedor para evaluar riesgos y cuantificar la resiliencia. El potencial de las BNs para mejorar la toma de decisiones en contextos complejos hace que sea crucial continuar explorando su uso. La literatura destaca múltiples estudios que exploran el uso de BNs en diferentes contextos. En el siguiente apartado de modelado usando redes bayesianas, se resume la revisión literaria de los artículos analizados en profundidad.

6.1 Modelado Usando Redes Bayesianas

Un evento disruptivo puede desencadenar reacciones en cadena que afectan a más de una sección de una red de suministro. En la actualidad se han utilizado las BNs para proporcionar un marco y evaluar los riesgos de suministro con precisión (Sharma, Routroy & Chanda, 2022).

Pai et al. (2003) fueron unos de los primeros investigadores en analizar los riesgos de la SC a través de BNs. Utilizando BNs establecieron factores de riesgo y límites de riesgo aceptables para activos de la cadena de suministro de trinitrotolueno (TNT) del Departamento de Defensa de EE. UU.

Lockamy & McCormack (2012) presentan una metodología en la que utilizando BNs modelan el riesgo de proveedores en cadenas de suministro, determinando la probabilidad de tres tipos de riesgo (externo, operativo y de red), y el posible impacto en los ingresos que un proveedor puede obtener en una organización.

En el año 2016, por primera vez las BNs fueron utilizadas en el contexto de la evaluación y selección de proveedores, modelando de manera efectiva las relaciones causales entre las variables. Hosseini & Barker (2016) proponen una BN para cuantificar la idoneidad de los proveedores en función de la resiliencia, de criterios primarios y de criterios ecológicos. Los autores demuestran la superioridad de las BNs en comparación con otras metodologías de cálculo y selección de proveedores. El documento también destacó las principales ventajas de usar un modelo de red bayesiana: flexibilidad de tipos de variables (por ejemplo, booleanas, discretas, continuas), análisis de inferencias y cuantificación de la incertidumbre.

Hosseini et al. (2016) mencionan que la literatura muestra que muchos de los factores que impulsan la resiliencia de un sistema son de naturaleza cualitativa, como la cooperación y el trabajo en equipo del personal durante disrupciones, o el grado de preparación frente a desastres naturales, entre otros. Pero que el reto en la investigación es cómo evaluarla de manera cuantitativa. Para cerrar esta brecha, implementan una red bayesiana para cuantificar la resiliencia de la cadena de suministro de un fabricante de ácido sulfúrico en Irán.

Ojha et al. (2018) usaron la teoría de BNs para estudiar los efectos en cascada de las interrupciones del suministro y las interrupciones simultáneas de redes de varios eslabones.

Bouzembrak & Marvins (2019) utilizan un enfoque de BN para identificar y cuantificar las relaciones entre los peligros de seguridad alimentaria, y factores climáticos, económicos y agronómicos. Lo anterior utilizando la información del Sistema de Alerta Rápida para Alimentos y Piensos (RASFF, Rapid Alert System for Food and Feed).

Hosseini & Ivanov (2019) basándose en los conceptos de Henry & Ramirez-Marquez, presentaron una nueva medida de resiliencia de la SC, que considera la propagación de interrupciones en el rendimiento de la SC (efecto dominó). Este efecto se manifiesta cuando una interrupción y su impacto asociado se propagan aguas abajo en la SC. En este sentido, miden la resiliencia de un OEM como una función de su vulnerabilidad y capacidad de recuperación cuando un proveedor no puede suministrar debido a una interrupción.

Rodgers & Singham (2019) consideran que las fallas en los estudios de ensayos clínicos causan interrupciones significativas en las operaciones de la SC, lo que genera ineficiencias operativas y pérdidas financieras. Por lo cual, presentaron un marco para construir una red bayesiana para cuantificar la probabilidad de una interrupción en una SC clínica.

Hosseini et al. (2020) realizaron un estudio cuyo objetivo fue desarrollar un marco conceptual y una medida para evaluar la resiliencia de la cadena de suministro en un entorno de sistema abierto o cadena de suministro que interactúa con su entorno externo, como proveedores, clientes y otras partes interesadas.

Liu et al. (2020) formulan un modelo de optimización de BN dinámica teniendo en cuenta la forma de la propagación del riesgo en un horizonte temporal. Consideran el hecho que el impacto de una interrupción a lo largo de la SC, conocido como efecto dominó, provoca que el riesgo de

interrupción de los socios ascendentes en la cadena se propague a los socios aguas abajo, con los correspondientes efectos adversos. Además, realizan un estudio de caso para demostrar la aplicabilidad del modelo propuesto y extraen algunas conclusiones organizacionales.

Chhimwal et al. (2021) proponen una metodología de BN para analizar cómo ocurre la propagación del riesgo en una cadena de suministro circular de una organización automotriz. A través de la revisión de literatura identificaron como principales tipos de riesgo los económicos, ambientales, sociales, tecnológicos, de gestión de residuos, la vulnerabilidad ante cambios rápidos y el riesgo de canibalización. Lo anterior para posteriormente estudiar el efecto de estos riesgos en el desempeño de la cadena de suministro circular, evaluando aspectos como pérdidas de ventas, el impacto en los ingresos de la cadena y los costos de almacenamiento de inventarios.

Hosseini & Ivanov (2021) desarrollan un modelo de BN que se puede utilizar para identificar detonantes de interrupciones de la SC y eventos de riesgo en medio de la pandemia de COVID-19, así como para cuantificar las consecuencias de interrupciones pandémicas. Los autores aprovechan las características de las BNs para simular y medir el impacto de diferentes factores detonantes en el rendimiento financiero y continuidad del negocio. Concluyen que los resultados se pueden utilizar como una herramienta de apoyo a la toma de decisiones para predecir y comprender mejor los impactos de la pandemia en el desempeño de la SC.

Librantz et al. (2021) mediante redes bayesianas y proceso de jerarquía analítica evalúan los riesgos principales en la cadena de suministro de software (SSC, Software Supply Chain), que incluyen la manipulación de productos durante el desarrollo o la entrega, la calidad, la garantía debido a defectos de software, retrasos en la producción y aumento de los costos de producción, los cuales se han venido convirtiendo en una preocupación creciente. Presentan un ejemplo numérico que clasifica a los proveedores de software según su nivel de riesgo.

Sakib et al. (2021) llevan a cabo un estudio en el que presentan un modelo de BN para predecir y evaluar desastres en la SC de petróleo y gas de EE. UU., basado en siete factores: técnico, económico, social, político, de seguridad, ambiental y legal. Sus resultados muestran que, de los siete factores responsables de los desastres de la cadena, los factores técnicos tienen el mayor impacto, mientras que los factores legales y políticos tienen el menor.

Sichani & Padgett (2021) ante la problemática de huracanes anteriores que han demostrado la vulnerabilidad de las cadenas de suministro de petróleo de Estados Unidos sometidas a estos eventos extremos, y la falta de modelos para evaluar el desempeño de la infraestructura de la cadena de suministro de petróleo ante huracanes. Presentan un marco probabilístico que utiliza redes bayesianas para la evaluación del desempeño de la infraestructura de la cadena de suministro de petróleo (OSCI, Oil Supply Chain Infrastructure) sometida a eventos de huracanes.

Badhotiya et al. (2022) proponen un modelo analítico para evaluar la resiliencia de la cadena de suministro y superar los efectos de los impactos de la disrupción. Los objetivos del estudio son: identificar y discutir sobre los indicadores de resiliencia de la cadena de suministro, modelar la interdependencia entre estos indicadores, evaluar la resiliencia de la cadena de suministro de tres empresas de la India en respuesta al brote de COVID-19 y realizar un análisis comparativo de los resultados. El enfoque propuesto puede ayudar a los responsables de la toma de decisiones a identificar los indicadores críticos en los cuales enfocarse para mejorar la resiliencia de la cadena, y superar el brote de la pandemia de COVID-19.

Çıkmak & Ungan (2022) crearon un modelo de BN para determinar las probabilidades de los riesgos y las estrategias de mitigación más efectivas para abordar los riesgos en la cadena de suministro automotriz. Se realizaron análisis de escenarios y de sensibilidad para evaluar la solidez

del modelo. La colaboración y el transporte flexible surgieron como las estrategias de mitigación más efectivas para abordar los riesgos en la red de suministro.

Fan et al. (2022) desarrollaron un nuevo modelo de riesgo basado en BNs para investigar el impacto de factores de riesgo en diferentes tipos de accidentes marítimos. Crearon una base de datos con 25 factores a partir de un análisis manual de accidentes registrados desde 2005 hasta 2021 en aguas restringidas importantes del mundo. El modelo es verificado mediante un análisis de sensibilidad y casos de accidentes reales.

Dada la gravedad de la congestión portuaria causada por diversos factores e intensificada con el COVID-19, la estabilidad de las cadenas de suministro internacionales se ha visto desafiada; por lo cual, Gui, Wang & Yu (2022) realizan un estudio sobre los riesgos de congestión portuaria durante la pandemia de COVID-19, desarrollando una metodología para la identificación y priorización del riesgo de congestión portuaria. Establecen un nuevo modelo de evaluación del riesgo de congestión y presentan un ejemplo ilustrativo en el que encuentran que los tres principales factores de riesgo son interrupción de los servicios ferroviarios/barcazas, la escasez de mano de obra calificada y la escasez de conductores de camiones de carga.

De otra parte, Liang et al. (2022) realizan un estudio para analizar los factores influyentes en el riesgo (RIFs, Risk Influential Factors) del robo de carga y predecir la probabilidad de diferentes tipos de accidentes relacionados. En el estudio se evalúan y priorizan los RIFs críticos que contribuyen al robo de carga con el fin de predecir la ocurrencia de posibles accidentes; para ello recopilaron datos históricos de 9316 accidentes de robo de carga que ocurrieron en el Reino Unido entre 2009 y 2021, obtenidos de la base de datos TAPA IIS, y luego construyeron un modelo de análisis de riesgo de robo de carga basado en BNs.

Liu et al. (2022a) utilizaron una BN para identificar los peligros químicos de seguridad alimentaria en la leche, asociados con una anomalía para los Países Bajos. Como caso de aplicación utilizan la SC de productos lácteos en Europa, detectando anomalías en los datos de indicadores que se esperaba impactaran en el desarrollo de riesgos de seguridad alimentaria en la leche. El modelo bayesiano fue construido en KNIME (Konstanz Information Miner), software de código abierto que permite realizar análisis de datos.

Liu et al. (2022b) investigan un nuevo problema de gestión de la propagación de disrupciones en una SC multinivel con un presupuesto limitado de intervención. El objetivo del estudio es minimizar el riesgo de disrupción medido por la probabilidad de disrupción de los participantes objetivo en la SC. Para ello, construyen dos modelos de programación no lineal entera-mixta para determinar las intervenciones apropiadas. El enfoque combina redes bayesianas, do-calculus y programación matemática. Realizan experimentos numéricos con instancias generadas aleatoriamente para evaluar la eficiencia de los modelos propuestos.

Ruskey & Rosenberg (2022) desarrollaron un enfoque para minimizar la demanda insatisfecha esperada en una cadena de suministro. Formulan el problema de seleccionar un conjunto óptimo de mejoras de nodos, minimizando la pérdida esperada en cadenas de suministro modeladas bayesianamente como un problema lineal entero binario (BIP, Binary Integer Program). Presentan resultados computacionales para problemas pequeños e ilustran cómo el conjunto de soluciones cambia con el presupuesto.

Sharma, Routroy & Chanda (2022) diseñan un modelo de riesgo de suministro que utiliza una BN cuyo objetivo es identificar los factores de riesgo más críticos que influyen en las redes de la SC. La metodología propuesta fue implementada a través de un estudio de caso en una industria de la India, encontrando que los tres principales factores que influyeron en la rentabilidad del

negocio fueron los retrasos, la tecnología del producto y el precio del combustible. La investigación tiene limitaciones como el no considerar un número más amplio de riesgos en el modelado.

Wang, Ding & Wang (2022) considerando que, en la actualidad, la mayoría de los países en desarrollo carecen de cadenas de suministro humanitarias (HSC, Humanitarian Supply Chains) resilientes y efectivas, desarrollan un modelo de BN para evaluar cuantitativamente el rendimiento de HSC en el contexto de la amenaza de ciclones tropicales en Zimbabue.

Bugert & Lasch (2023) proponen una combinación de modelado basado en agentes y redes bayesianas para el análisis de riesgos en redes de suministro. Presentan una metodología que captura las interrelaciones complejas de varios escenarios de riesgo, ofreciendo una visión de las propagaciones tanto hacia adelante como hacia atrás. Además, proporcionan tres métricas para redes bayesianas que cuantifican la propagación del riesgo. El enfoque se ilustra mediante su aplicación a una red de suministro de la industria de bienes de consumo.

Zhou et al. (2023) investigan los riesgos en la cadena de suministro de cruceros de Shanghai. Identificaron y evaluaron varios factores de riesgo, como la congestión del puerto, los problemas de seguridad, la gestión ineficiente de la cadena de suministro y los eventos climáticos adversos. Los hallazgos y resultados del estudio pueden ayudar a las autoridades portuarias, las compañías de cruceros y otros actores relevantes a comprender mejor los riesgos y tomar medidas preventivas o correctivas para mitigarlos.

En este apartado, se realizó una revisión de literatura para identificar trabajos relevantes que abordan riesgos y resiliencia en la cadena de suministro, con énfasis particular en el uso de redes bayesianas. La Tabla 4 presenta un resumen de los estudios seleccionados, incluyendo la descripción del estudio, su enfoque principal y si combina BNs con otras metodologías.

Tabla 4.

Resumen de revisión modelado usando redes bayesianas

Referencia	Descripción	Enfoque		Combinan
		Riesgos	Resiliencia	
Pai et al. (2003)	Factores y límites de riesgo para activos de la cadena de suministro	X		
Lockamy & McCormack (2012)	Metodología para modelar el riesgo de proveedores en cadenas de suministro.	X		
Hosseini y Barker (2016)	Cuantifican la idoneidad de proveedores	X		
Hosseini et al. (2016)	Cuantifican la resiliencia de la cadena de suministro de un fabricante de ácido sulfúrico en Irán.		X	
Ojha et al. (2018)	Efectos en cascada de las interrupciones del suministro y simultáneas de redes de varios eslabones.		X	
Bouzembrak & Marvins (2019)	Identifican y cuantifican las relaciones entre los peligros de seguridad alimentaria, y factores climáticos, económicos y agronómicos.	X		
Hosseini & Ivanov (2019)	Presentan una nueva medida de resiliencia de la SC, que considera la propagación de interrupciones en su rendimiento (efecto dominó).		X	
Rodgers & Singham (2019)	Marco para construir una red bayesiana que cuantifica la probabilidad de una interrupción en una cadena de suministro clínica.	X		
Hosseini et al. (2020)	Marco conceptual y una medida para evaluar la resiliencia de la cadena de suministro en un entorno de sistema abierto.		X	
Liu et al. (2020)	Formulan un modelo de optimización de BN dinámica teniendo en cuenta la forma de la propagación del riesgo en un horizonte temporal	X		Prog. Matemática
Chhimwal et al. (2021)	Metodología de BN para analizar cómo ocurre la propagación del riesgo en una red de suministro circular de una automotriz.	X		
Hosseini & Ivanov (2021)	Identifican detonantes de interrupciones de la SC y eventos de riesgo en pandemia de COVID-19, cuantificando consecuencias.	X		
Librantz et al. (2021)	Mediante redes bayesianas y proceso de jerarquía analítica evalúan los riesgos principales en la cadena de suministro de software (SSC).	X		Proceso de jerarquía analítica
Sakib et al. (2021)	Modelo de BN para predecir y evaluar desastres en la SC de petróleo y gas de EE. UU.	X		
Sichani & Padgett (2021)	Marco para evaluar el desempeño de la SC de petróleo (OSCI, Oil Supply Chain Infrastructure) sometida a huracanes.	X		
Badhotiya et al. (2022)	Modelo analítico para evaluar la resiliencia de la cadena de suministro y superar los efectos de los impactos de la disrupción.		X	
Çıkmak & Ungan (2022)	Modelo para determinar probabilidades de los riesgos y estrategias de mitigación efectivas para abordar los riesgos en la cadena automotriz.	X		
Fan et al. (2022)	Modelo de riesgo basado en BNs para investigar el impacto de factores de riesgo en diferentes tipos de accidentes marítimos	X		
Gui, Wang & Yu (2022)	Nuevo modelo de evaluación del riesgo de congestión portuaria.	X		
Liang et al. (2022)	Analizan los factores influyentes en el riesgo del robo de carga y predecir la probabilidad de diferentes tipos de accidentes relacionados.	X		
Liu et al. (2022a)	Identifican los peligros químicos de seguridad alimentaria en la leche, asociados con una anomalía para los Países Bajos.	X		
Liu et al. (2022b)	Modelos de programación no lineal entera-mixta para determinar las intervenciones apropiadas en una SC multinivel con presupuesto limitado. Combinan BN, do-calculus y programación matemática.	X		Cálculo do y Prog. Matemática
Ruskey & Rosenberg (2022)	Problema de seleccionar un conjunto óptimo de mejoras de nodos, minimizando la pérdida esperada en cadenas de suministro modeladas bayesianamente como un problema lineal entero binario.	X		Prog. matemática
Sharma, Routroy & Chanda (2022)	Modelo de riesgo de suministro cuyo objetivo es identificar los factores de riesgo más críticos que influyen en las redes de la SC.	X		
Wang, Ding & Wang (2022)	Evalúan el rendimiento de cadenas de suministro humanitarias en el contexto de la amenaza de ciclones tropicales en Zimbabue.	X		
Bugert & Lasch (2023)	Combinan modelado basado en agentes y BNs para análisis de riesgos en redes de suministro. Consideran propagación en ambas direcciones.	X		Modelado Agentes
Zhou et al. (2023)	Investigan los riesgos en la SC de cruceros de Shanghái.	X		

7. Modelo de Red Bayesiana para el Análisis de Riesgos y Resiliencia de la Cadena de Suministro

7.1 Modelo Conceptual

Los sistemas complejos se caracterizan por la presencia de múltiples aspectos interrelacionados (Koller & Friedman, 2009). Una SC puede considerarse como un sistema complejo debido a la interrelación entre sus múltiples actores, como proveedores, fabricantes, mayoristas y minoristas. El estado de interrupción de uno de ellos puede impactar a los siguientes. Por ejemplo, si un proveedor experimenta una interrupción en su operación, los fabricantes que dependen de él pueden verse afectados, lo que, repercutirá en los mayoristas y minoristas que necesitan esos productos para satisfacer la demanda. Así, una interrupción en un punto de la cadena puede propagar efectos adversos a lo largo de la red, lo que resalta la importancia de modelar y comprender estas interacciones para analizar los riesgos y la resiliencia del sistema.

El modelo conceptual que se propone parte de la simulación del funcionamiento de la SC, tomando como referencia la estructura de la red de suministro presentada por Bugert & Lasch (2023), quienes abordan un caso en la industria de bienes de consumo. En su estudio, combinan modelado basado en agentes y las redes bayesianas para el análisis de riesgos de la red de suministro. El modelado basado en agentes es una técnica utilizada en sistemas complejos, que “se basa en modelar los sistemas desde el punto de vista de los elementos que los componen; los agentes, y las relaciones entre ellos”. (Pereda & Zamarreno, 2015).

Bugert & Lasch (2023) consideran una estructura de red de suministro compuesta por 10 agentes (actores de la cadena), modelando las relaciones de flujo de materiales, los procesos de producción con tiempos de proceso estocásticos y tiempos de transporte interorganizacionales también estocásticos. Abordan el comportamiento de compra y demanda, los niveles de inventario

y las modalidades de entrega. Tienen en cuenta los costos asociados al almacenamiento, la producción y las interrupciones. Sin embargo, no especifican qué herramienta utilizan para la simulación basada en agentes.

En cuanto al modelado mediante redes bayesianas, el conjunto de nodos de la red corresponde a la estructura de la red de suministro, y consideran 12 escenarios de riesgo. Los primeros 10 escenarios contemplan la interrupción de un solo actor de la red, mientras que los dos restantes consideran interrupciones simultáneas en dos actores. Para cada escenario, se asume una probabilidad de ocurrencia y se crea una red bayesiana individual, cuyas probabilidades condicionales se calculan en función de la gravedad de la interrupción, medida en días, lo que implica que los estados de los nodos no son binarios. Las redes bayesianas se modelan y analizan utilizando las herramientas de software GeNie Modeler y SMILE Engine. El enfoque del estudio se centra en el análisis de riesgos, para lo cual presentan tres métricas nuevas que abordan la propagación del riesgo. No tratan la cuantificación de la resiliencia de la cadena de suministro.

A partir de este artículo base, se construye el modelo conceptual que se propone a continuación, en donde se omiten algunos datos y supuestos como, la propagación del riesgo aguas arriba, los costos de almacenamiento, producción e interrupción. Otros han sido adaptados, como el tipo de política de inventario utilizada, los cálculos de demanda y el calculo de probabilidades condicionales para el modelado bayesiano en el cual los autores en mención plantean los escenarios de riesgo y, para cada uno de ellos, crean una BN. Por el contrario, en el modelo conceptual que se propone en este trabajo, las probabilidades condicionales se determinan según el registro del historial de días con falta de stock, construyendo una red bayesiana única. Los eventos de riesgo se consideran mediante la introducción de evidencia en la red bayesiana resultante.

Por lo expuesto, la dinámica de las operaciones internas de los eslabones difiere, y la operación de la cadena final es una cadena hipotética. No obstante, la estructura general de la red y las interrelaciones entre los actores de la cadena de suministro se mantienen sin modificaciones, dado que los eslabones siguen el orden definido.

La estructura de la red es la que se presenta en la Figura 5, y está compuesta por 10 actores o participantes: $S1$, $S2$, $S3$, $P1$, $P2$, $W1$, $W2$, $R1$, $R2$, y $R3$. Se consideran los procesos de producción con tiempos de proceso estocásticos para los actores que realizan manufactura (proveedores y productores), así como tiempos de entrega estocásticos, y comportamiento incierto de la demanda. Además, se implementa la política de inventario (s, S) para cada participante, teniendo en cuenta el seguimiento de los niveles de inventario e inventario de seguridad. En el apéndice B se presentan conceptos asociados a la gestión de inventarios y a la política (s, S).

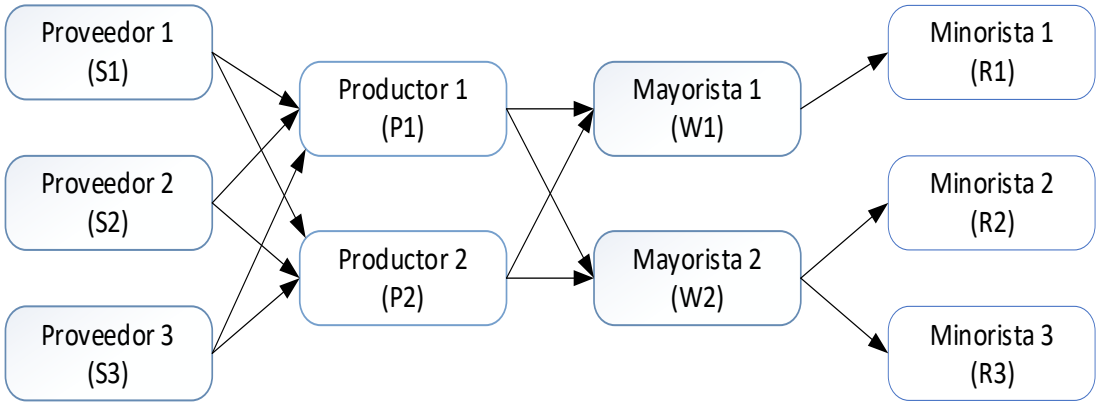
La configuración de las relaciones entre los diferentes participantes o la dirección del flujo de bienes es la siguiente:

- $R1$ recibe productos de $W1$, mientras que $R2$ y $R3$ reciben de $W2$.
- $W1$ recibe productos de $P1$ y $P2$, y de manera similar, $W2$ también recibe de $P1$ y $P2$.
- $P1$ y $P2$ reciben material de $S1$, $S2$, y $S3$, es decir, $S1$**,** $S2$**, y** $S3$ suministran tanto a $P1$ como a $P2$**.**

Estas interacciones muestran cómo la cadena no se limita a un único actor en cada etapa, sino que representa una red compleja de relaciones. Por ejemplo, cada fabricante, en este caso Productor 1 y Productor 2, recibe material de tres proveedores y abastece a dos mayoristas, que a su vez suministran el producto terminado a los minoristas: $W1$ abastece a $R1$, y $W2$ a $R2$ y $R3$ (Bugert & Lasch, 2023).

Figura 5.

Estructura de la cadena de suministro



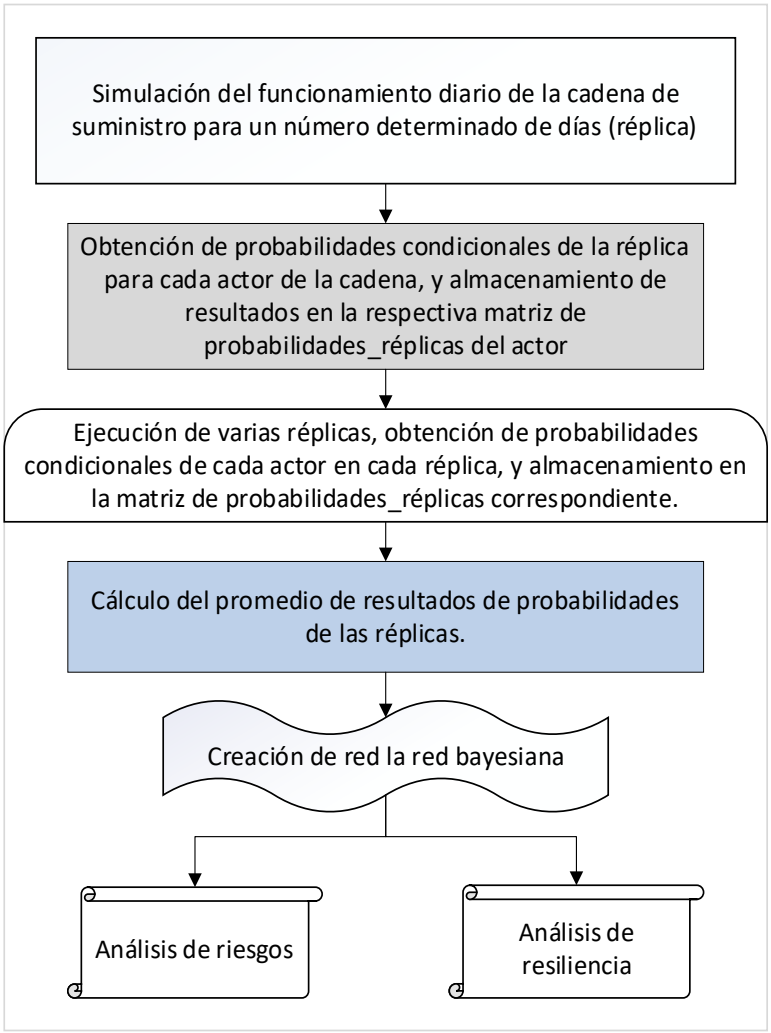
Adaptado de "*Analyzing upstream and downstream risk propagation in supply networks by combining Agent-based Modeling and Bayesian networks*", por N. Bugert & R. Lasch, 2023, Journal of Business Economics.

La dinámica de funcionamiento de la cadena se simula durante 1800 días (periodo de duración de una réplica) con el fin de generar la data que permita detectar la propagación del riesgo y obtener las probabilidades condicionales preliminares de las variables que componen la BN. Para garantizar que las propiedades estocásticas del sistema se reflejen de manera realista en los resultados, se ejecutan 5 réplicas y se calculan las probabilidades condicionales, que se utilizarán como entrada en la creación de la BN. La estructura de la red se basa en las relaciones de flujo de materiales o productos entre los participantes de la cadena, reflejando sus complejas interacciones.

A partir de la BN creada, se realiza el análisis de riesgos y el de resiliencia utilizando una métrica propuesta. No se contempla la propagación de riesgos aguas arriba.

Figura 6.

Modelo conceptual



El modelo conceptual descrito se sintetiza en la Figura 6. Su implementación se lleva a cabo en Python, utilizando el entorno de programación Google Colab, que proporciona una interfaz robusta y útil para la simulación y los análisis correspondientes.

7.1.1 Supuestos de la dinámica de funcionamiento de la cadena (Adaptado de Bugert & Lasch, 2023).

- Los proveedores ($S1$, $S2$, $S3$) registran una recepción de materia prima después de un tiempo de entrega inmediato, definido en 0.

- Los proveedores ($S1$, $S2$, $S3$) proporcionan la materia prima con cierto proceso de manufactura (materia prima procesada) para que cada productor elabore producto terminado.
- Solo los productores ($P1, P2$) y los proveedores ($S1$, $S2$, $S3$) llevan a cabo proceso de manufactura (proceso de producción).
- Los proveedores transforman la materia prima, y el proceso de producción comienza tan pronto como esta se encuentre disponible.
- Para fabricar un producto, cada productor necesita una unidad de materia prima de cada proveedor. El proceso de producción comienza tan pronto como los tres materiales están disponibles.
- Las duraciones de los procesos de producción siguen una distribución uniforme, y se tiene en cuenta la capacidad máxima de producción por período (día simulado). Esto solo aplica a proveedores y los productores, ya que son los únicos que tienen proceso de manufactura.
- El inventario para proveedores y productores se clasifica en: producto terminado, en proceso y materia prima. En el caso de los mayoristas y minoristas solo tienen producto terminado.
- El número de clientes por día (demanda) sigue una distribución normal. Se compran los productos si están disponibles. En caso de desabastecimiento, el cliente espera al minorista un tiempo limitado (máximo un día), modelado por una distribución normal.
- Los mayoristas ($W1$, $W2$) compran el producto terminado a los productores ($P1$, $P2$), siendo el producto el mismo para ambos. Cada mayorista distribuye su orden de pedido de manera equitativa entre los dos productores, la mitad de cada orden se envía a $P1$ y la otra mitad a $P2$.

- En el caso de los proveedores, productores y mayoristas revisan las ordenes de pedido pendientes frente al inventario disponible y, si es posible, despachan todo el volumen solicitado. Si el stock es insuficiente, se realiza un cumplimiento parcial y el pedido se completa cuando el stock esté disponible. Se lleva un registro de la cantidad de productos despachados y el atraso de estos. En cuanto a los minoristas, también de acuerdo con el inventario pueden despachar toda la cantidad solicita por el cliente, o hacer un despacho parcial; sin embargo, como se mencionó, el cliente solo está dispuesto a esperar 1 día por la cantidad restante.
- La cantidad despachada se entrega después de un tiempo de entrega que sigue una distribución uniforme. Los minoristas entregan el pedido solicitado de manera inmediata.
- El volumen de productos entrantes de cada participante se ajusta cada vez que se reciben bienes o productos para llevar control de las cantidades pendientes de recibir.
- Se registra cada día que los participantes de la cadena carecen de inventario, con el fin de calcular las probabilidades condicionales.

  Para registrar cada día que un participante carece de inventario, se define una variable booleana (dia_sin_stock) que rastrea si éste tiene suficiente stock para procesar todas las órdenes del día. Esta variable es inicializada como False (self.dia_sin_stock = False), lo que significa que no hay falta de stock en el participante. Y para cada orden de la lista de ordenes pendientes:

a) Si la cantidad solicitada por el cliente es menor o igual al stock en existencias (hay suficiente stock), se procesa completamente la orden.
b) Si hay algo de stock, pero no lo suficiente para satisfacer toda la orden (no hay suficiente stock) se procesa parcialmente.

- Se despacha la cantidad disponible, y la orden original se modifica para reflejar solo la cantidad despachada. Se cambia el estado a PROCESADA.
- Se crea una nueva orden (orden_restante) que contiene la parte que no pudo despacharse. La nueva orden restante se inserta en la lista de pendientes justo después de la orden actual.
- La bandera dia_sin_stock se marca como True, indicando que no hubo suficiente stock en ese día.

c) Si no hay stock: la orden no se procesa y la bandera dia_sin_stock se marca como True.

d) Se actualiza un historial (historial_dias_sin_stock), agregando si en el día analizado hubo o no falta de stock. Por ejemplo, si se simularon 3 días, y el historial de días sin stock de $W1$ es [False, False, False], quiere decir que no se detectó en ninguno de los 3 días falta de stock en el participante $W1$.

- Tras despachar todos los pedidos posibles, el inventario se monitorea diariamente al final del día, para determinar si se debe realizar un nuevo pedido (para ello se usa la política de pedido (s, S)). Cada participante, verifica su nivel objetivo de stock para calcular la cantidad de pedido a su respectivo proveedor. Las órdenes de pedidos resultantes se envían inmediatamente.

7.1.2 Parámetros de entrada

Los parámetros de entrada utilizados en la simulación son los siguientes (Bugert & Lasch, 2023):

Demanda y desviación estándar de cada participante:

El número de consumidores diarios interesados en comprar un artículo sigue una distribución normal, con una media de 250 unidades diarias y una desviación estándar de 35

unidades. En el presente trabajo se asume que esta distribución corresponde a la demanda diaria individual de cada minorista ($R1$, $R2$ y $R3$) dado que el artículo de Bugert & Lasch (2023) no especifica cómo se distribuye entre los actores de la cadena. A partir de estos valores (media de 250 y desviación estándar de 35), y dado tampoco se especifican dependencias entre las mismas, se asume que son independientes entre sí. Con base en estos supuestos, se calculan las demandas y desviaciones estándar para los demás actores de la cadena de suministro, siguiendo el procedimiento descrito a continuación:

Cálculo de la demanda y desviación estándar para cada minorista:

- **Media para cada minorista $R1$, $R2$ y $R3$:** La demanda de cada minorista es 250 unidades diarias: $\boldsymbol{\mu_{R1} = \mu_{R2} = \mu_{R3} = 250}$
- **Desviación estándar por minorista:** La desviación estándar de la demanda para cada minorista es 35 unidades: $\boldsymbol{\sigma_{R1} = \sigma_{R2} = \sigma_{R3} = 35}$

Cálculo de la demanda y desviación estándar de cada mayorista, considerando los participantes a quienes abastece:

Para $W1$: El minorista $R1$ es el único cliente del mayorista $W1$, y $R1$ genera todas sus órdenes de compra exclusivamente a $W1$. Por lo tanto, la varianza y la desviación estándar de $W1$ serán idénticas a las de $R1$, absorbiendo toda la variabilidad de la demanda del minorista.

$$\boldsymbol{\mu_{W1} = \mu_{R1} = 250} \quad ; \quad \boldsymbol{\sigma_{W1} = \sigma_{R1} = 35}$$

Para $W2$: En el caso de $W2$, se tiene que:

$$\mu_{W2} = \mu_{R2} + \mu_{R3} \quad ; \quad \boldsymbol{\mu_{W2}} = 250 + 250 = \boldsymbol{500}$$

La varianza de la demanda combinada que enfrenta $W2$ se calcula sumando las varianzas individuales de los clientes a quienes abastece. Por lo tanto, la varianza total que enfrenta $W2$ es la suma de las varianzas de las demandas de $R2$ y $R3$:

$$Varianza_{W2} = Varianza_{R2} + Varianza_{R3} \quad ; \quad \boldsymbol{\sigma_{W2}^2} = 35^2 + 35^2 = \mathbf{2450}$$

Luego, la desviación estándar de $W2$ es:

$$\boldsymbol{\sigma_{W2}} = \sqrt{\mathbf{2450}} \approx \mathbf{49.50}$$

Cálculo de la demanda y desviación estándar de cada productor, considerando los participantes a quienes abastece:

Para $P1$: Cada mayorista, $W1$ y $W2$, distribuye su demanda entre $P1$ y $P2$, envían el 50% de su demanda a $P1$ y el otro 50% a $P2$, luego $P1$ recibe la mitad de la demanda de $W1$ y la mitad de la demanda de $W2$. En este contexto, la demanda total que enfrenta cada productor es la suma de la mitad de la demanda de cada mayorista al que suministra. Así, la demanda media de $P1$ es:

$$\mu_{P1} = \frac{1}{2}\mu_{W1} + \frac{1}{2}\mu_{W2} \quad ; \quad \boldsymbol{\mu_{P1}} = \frac{1}{2}(250) + \frac{1}{2}(500) = \mathbf{375}$$

La varianza total que enfrenta cada productor se calcula sumando las varianzas ponderadas de las demandas que recibe de cada mayorista. Cada productor recibe la mitad de la demanda de cada mayorista; por lo tanto, la varianza total que enfrenta el productor $P1$ es:

$$\sigma_{P1}{}^2 = \left(\frac{1}{2}\sigma_{W1}\right)^2 + \left(\frac{1}{2}\sigma_{W2}\right)^2 \quad ; \quad \boldsymbol{\sigma_{P1}^2} = \frac{1}{4}(35^2) + \frac{1}{4}(2450) = \mathbf{918.75}$$

Luego, la desviación estándar de $P1$ es:

$$\boldsymbol{\sigma_{P1}} = \sqrt{\mathbf{918.75}} \approx \mathbf{30.31}$$

Para $P2$: Dado que $P2$ recibe la misma proporción de la demanda de $W1$ y $W2$, la media y la desviacion estándar de la demanda para $P2$ son iguales a las de $P1$:

$$\boldsymbol{\mu_{P2}} = \mathbf{375} \quad ; \quad \boldsymbol{\sigma_{P2}} = \sqrt{\mathbf{918.75}} \approx \mathbf{30.31}$$

Cálculo de la demanda y desviación estándar de cada proveedor, considerando los participantes a quienes abastece:

Los proveedores $S1$, $S2$ y $S3$ enfrentan una demanda proporcional a la demanda total de productos que deben producir los productores $P1$ y $P2$. La demanda de cada proveedor se calcula de la misma manera ya que ambos productores necesitan la misma cantidad de materia prima de cada proveedor. Cada productor ($P1$ y $P2$) necesita una unidad de materia prima de cada proveedor para fabricar una unidad de producto.

La demanda total de cada proveedor es la suma de la demanda de $P1$ y $P2$

$$\mu_{S1} = \mu_{S2} = \mu_{S3} \ = \ \mu_{P1} + \mu_{P2} = 375 + 375$$

$$\boldsymbol{\mu_{S1} = \mu_{S2} = \mu_{S3} \ = 750}$$

La varianza de la demanda total enfrentada por los proveedores es:

$$\sigma_{S1}{}^2 = \sigma_{S2}{}^2 = \sigma_{S3}{}^2 = \ \sigma_{P1}{}^2 + \sigma_{P2}{}^2 \ = \ \boldsymbol{918.75 \ + \ 918.75}$$

$$\sigma_{S1}{}^2 = \sigma_{S2}{}^2 = \sigma_{S3}{}^2 = \ 1837.5$$

La desviación estándar es:

$$\boldsymbol{\sigma_{S1} = \sigma_{S2} = \sigma_{S3} = \sqrt{1837.5} \ \approx \ 42.87}$$

La Tabla 5 presenta un resumen de los parámetros utilizados para la inicialización de instancias, incluyendo la media y desviación estándar de la demanda de cada participante. El tiempo de entrega (inmediato del minorista al cliente, y variable para los demás participantes, siguiendo una distribución uniforme entre el mínimo y máximo especificado). Se incluyen además los tiempos de producción (también distribuidos uniformemente, oscilando entre 4 y 6 días para los productores y entre 6 y 10 días para los proveedores). Los niveles de stock objetivo para efectos de la política de inventario (específicos para cada participante). El stock en proceso (establecido en cero). El stock inicial PROCESADO/TERMINADO (específico para cada participante). Por último, el límite máximo de producción de 20,000 piezas por día para proveedores y productores

(en el caso de los demás participantes, este límite no se establece, ya que, como se mencionó, no tienen proceso de producción).

Tabla 5.

Resumen de parámetros de inicialización de instancias

	S1	S2	S3	P1	P2	W1	W2	R1	R2	R3
Demanda media [unidades]	**750**	**750**	**750**	**375**	**375**	**250**	**500**	**250**	**250**	**250**
Desviación estándar [unidades]	$\sqrt{1837.5}$	$\sqrt{1837.5}$	$\sqrt{1837.5}$	$\sqrt{918.75}$	$\sqrt{918.75}$	**35**	$\sqrt{2450}$	**35**	**35**	**35**
Tiempo mínimo de entrega [días]	4	4	4	3	3	1	1	0	0	0
Tiempo máximo de entrega [días]	8	8	8	5	5	3	3	0	0	0
Tiempo mínimo de producción [días]	6	6	6	4	4	0	0	0	0	0
Tiempo máximo de producción [días]	10	10	10	6	6	0	0	0	0	0
Nivel objetivo [unidades]	11000	11000	12000	3000	4000	2000	4000	1500	1500	1500
Stock en proceso [unidades]	0	0	0	0	0	0	0	0	0	0
Stock inicial PROCESADO /TERMINADO [unidades]	11000	11000	12000	3000	4000	2000	4000	1500	1500	1500
Límite máximo de producción al día [unidades]	20000	20000	20000	20000	20000	0	0	0	0	0

Adaptado de *"Analyzing upstream and downstream risk propagation in supply networks by combining Agent-based Modeling and Bayesian networks"*, por N. Bugert & R. Lasch, 2023, *Journal of Business Economics*.

7.1.3 Creación de clases/etapas

En programación orientada a objetos (OOP, Object-Oriented Programming), se tienen dos conceptos fundamentales: objetos y clases, donde las clases actúan como plantillas para la creación de objetos. Estas clases definen los atributos (características) y los métodos (comportamientos o funciones) que los objetos creados a partir de ellas tendrán (Schildt, 2011). Los objetos generados a partir de una clase incorporan las propiedades y comportamientos establecidos por esta, aunque no todos los métodos se utilicen en cada instancia.

En ese orden de ideas, para el modelado en Python, se definieron cuatro clases que representan las diferentes etapas de la SC: proveedor, productor, mayorista y minorista. La definición de cada clase abarca desde el constructor (método especial), hasta los métodos específicos de funcionamiento de la clase. El constructor se encarga de inicializar los atributos del objeto cuando se crea una instancia de esa clase, así como de definir otros atributos que cualquier instancia de la clase pueda tener.

Los participantes de la SC se representan como objetos: $R1$, $R2$ y $R3$ son instancias de la clase minorista; $W1$ y $W2$ de la clase mayorista; $P1$ y $P2$ de la clase productor; y $S1$, $S2$ y $S3$ de la clase proveedor. Cada objeto es una instancia de su respectiva clase, lo que significa que cada participante de la cadena tiene atributos y comportamientos propios de la clase a la que pertenece.

7.1.4 Métodos de cada clase

Para cada clase se definen sus respectivos métodos. En el apéndice C se detalla el funcionamiento de cada método asociado a las clases minorista, mayorista, productor y proveedor, los cuales fueron implementados en Python. A manera de resumen, los métodos de cada clase incluyen el constructor y otros relacionados con las operaciones diarias de los participantes, como el proceso de producción (determinando la cantidad a producir según capacidad e inventario -esto

para las clases que manufacturan-), la recepción de órdenes de compra y su procesamiento (despacho de órdenes pendientes y del día, haciendo un rastreo del historial de días sin stock), el cálculo de la cantidad a ordenar a quien le abastece (considerando el punto de reorden, la incertidumbre en la demanda y los tiempos de entrega y procesamiento), la generación de la orden de compra, y la recepción de los bienes o productos con la actualización del inventario entrante.

7.1.5 Creación de instancias

Cada instancia se crea con los parámetros definidos en el constructor, que en Python corresponde al método __init__. Por ejemplo, en la clase proveedor, los parámetros son:

- **self:** Hace referencia a la instancia actual de la clase, permitiendo asignar valores a las propiedades del objeto, llamar a otros métodos dentro de la clase y gestionar atributos y comportamientos específicos de la instancia.
- **nombre:** Parámetro que asigna un valor específico al atributo self.nombre de la instancia.
- **media_demanda y desv_demanda:** Media y desviación estándar de la demanda esperada.
- **min_tiempo_entrega y max_tiempo_entrega:** El tiempo mínimo y máximo de entrega.
- **min_tiempo_pcc y max_tiempo_pcc:** El tiempo mínimo y máximo de producción
- **nivel_objetivo:** El nivel objetivo de inventario PROCESADO/TERMINADO que se desea mantener para el participante.
- **stock_material_en_proceso:** La cantidad de material en proceso
- **stock_material_provee_PROCESADO:** La cantidad de materia prima/producto ya procesado.
- **max_limite_pcc:** La capacidad máxima de producción por día.

En el caso de las demás clases, los parámetros son los similares, aunque con modificaciones en los nombres para adaptarlos a cada clase. Por ejemplo, en la clase productor,

stock_material_en_proceso se renombra a stock_producto_en_proceso_PRODU; y lo mismo ocurre en las clases mayorista y minorista, donde se ajustan los nombres a su respectivo contexto.

7.1.6 Dinámica del funcionamiento diario de la cadena

La Figura 7 muestra el pseudocódigo que resume el flujo de funcionamiento de la SC.

Figura 7.

Pseudocódigo de funcionamiento de la cadena de suministro

```
// Imprimir encabezado("******MINORISTA******")
// Procesar venta pendiente de los minoristas R1, R2, R3
para cada minorista (R1, R2, R3):
    llamar a procesar_venta_pendiente()

// Imprimir_encabezado("Colocación de pedidos del Minorista al Mayorista")

// Calcular y generar órdenes de compra a los mayoristas
para cada minorista (R1, R2, R3):
    demandaMino = llamar a demanda_venta()
    candidad_orden = llamar a calculo_cantidad_a_ordenar_al_mayorista()
    si cantidad_orden > 0 entonces
        llamar a (cantidad_orden, "Wx")
    si no
        imprimir("minorista" no coloca una orden al mayorista "Wx" en este
día)

// Imprimir encabezado("******MAYORISTA******")
// Procesar órdenes pendientes para cada mayorista
para cada mayorista (W1, W2):
    llamar a `mayorista_procesa_ordenes_pendientes_del_minorista()`

// Imprimir_encabezado("Colocación de pedidos del Mayorista a los
Productores")

// Calcular y generar órdenes de compra a los productores por parte de W1
`W1cantidad_orden = llamada a calculo_cantidad_a_ordenar_al_productor()`
si W1cantidad_orden > 0:
     cantidad_a_ordenar_mayo_p1 = W1cantidad_orden * 0.5
     cantidad_a_ordenar_mayo_p2 = W1cantidad_orden * 0.5
     llamada a generar_orden_al_productor(cantidad_a_ordenar_mayo_p1,
cantidad_a_ordenar_mayo_p2)
sino:
    imprimir "W1 no coloca órdenes a los productores"
```

Figura 7. (continuación)

Pseudocódigo de funcionamiento de la cadena de suministro (continuación)

```
// Colocar órdenes de compra a los productores por parte de W2
`W2cantidad_orden = llamada a calculo_cantidad_a_ordenar_al_productor()`
si W2cantidad_orden > 0:
    cantidad_a_ordenar_mayo_p1 = W2cantidad_orden * 0.5
    cantidad_a_ordenar_mayo_p2 = W2cantidad_orden * 0.5
    llamada a generar_orden_al_productor(cantidad_a_ordenar_mayo_p1,
cantidad_a_ordenar_mayo_p2)
sino:
    imprimir "W2 no coloca órdenes a los productores"

// Imprimir encabezado("******PRODUCTOR******")

imprimir_encabezado("Proceso de Producción del Productor")

// Proceso de producción para cada productor
para cada productor (P1, P2):
    llamar a `produccion_PRODU()`

// Imprimir encabezado("Colocación de pedidos del Productor a los
Proveedores")

// Procesar órdenes pendientes para cada productor
para cada productor (P1, P2):
    llamar a `productor_procesa_ordenes_pendientes_del_mayorista()`

// Calcular y generar órdenes a los proveedores
para cada productor (P1, P2):
    `demanda_Produ = llamada a demanda_venta_PRODU()`
    para cada Proveedor (S1, S2, S3):
        `cantidad_ordenSx = llamada a
calculo_cantidad_a_ordenar_al_proveedorSx()`
        imprimir: "Resultado de calculo_cantidad_a_ordenar_al_proveedor Sx
por parte de Px:", `cantidad_ordenSx`

    llamar a `generar_orden_al_proveedor(cantidad_ordenS1,
cantidad_ordenS2, cantidad_ordenS3)`
```

Figura 7. (continuación)

Pseudocódigo de funcionamiento de la cadena de suministro (continuación)

```
// Imprimir encabezado("******PROVEEDOR******")

// Imprimir_encabezado("Proceso de producción del Proveedor")

// Proceso de producción para cada proveedor
para cada proveedor (S1, S2, S3):
    llamar a `produccion_PROVEE()`

// Imprimir encabezado("Necesidades del Proveedor")

// Procesar órdenes pendientes para cada proveedor
para cada proveedor (S1, S2, S3):
    llamar a `proveedor_procesa_ordenes_pendientes_del_productor()`

// Calcular y generar órdenes de materia prima
para cada proveedor (S1, S2, S3):
    `cantidad_orden = llamada a calculo_cantidad_necesaria_mp()`
    imprimir: "Resultado de calculo_cantidad_necesaria por parte de Sx:",
`cantidad_orden`
    si `cantidad_orden > 0`:
        llamar a `generar_orden_mp(cantidad_orden)`
    sino:
        imprimir: "Sx no necesita mp"

// Procesar ordenes de materia prima para proveedores S1, S2, y S3
para cada proveedor (S1, S2, S3):
    llamar a `procesar_orden_mp()`
```

7.2 Obtención de Probabilidades Condicionales

Para construir la BN, es esencial contar con las probabilidades condicionales de todos los participantes de la cadena $S1$, $S2$, $S3$, $P1$, $P2$, $W1$, $W2$, $R1$, $R2$ y $R3$. Cada participante puede estar en dos estados: 0 indica el estado "Operativo" y 1 indica el estado "No operativo".

Los estados se asignan en función de si el participante tiene pedidos pendientes y carece de stock para cumplirlos. Si un participante no tiene stock y tiene pedidos pendientes por despachar, se le asigna el estado 1 (no operativo); en caso contrario, se le asigna el estado 0 (operativo). La

simulación del funcionamiento de la cadena permite hacer un seguimiento de los estados de cada actor durante los 1800 días de la réplica. Cada día que un participante carece de inventario se actualiza en la variable booleana *dia_sin_stock* y se lleva un registro de estos valores para los 1800 días simulados en cada réplica, los cuales se almacenan en la lista *historial_dias_sin_stock*. Con esta información, es posible calcular las probabilidades condicionales.

7.2.1 Probabilidades condicionales para cada actor para cada réplica de simulación

En el caso de los proveedores $S1$, $S2$ y $S3$.

Los proveedores $S1$, $S2$ y $S3$ no tienen predecesores, por lo que sus probabilidades se calculan directamente en función de su historial de estado. Al no depender de otros proveedores, las probabilidades de que cada uno esté operativo o no, se determinan sin considerar condiciones adicionales. En este sentido, estas probabilidades no son condicionales, sino marginales, ya que describen la probabilidad de los eventos individuales de $S1$, $S2$ y $S3$ sin referencia a otros eventos.

Para determinar la probabilidad marginal de que $S1$, $S2$ y $S3$ se encuentren en cada uno de sus dos estados (0 o 1) se sigue el siguiente procedimiento:

1. Se establecen e inicializan en cero los contadores (total_S1_op, total_S1_no_op, total_S2_op, total_S2_no_op, total_S3_op, total_S3_no_op) para registrar el total de días en los que cada proveedor ($S1$, $S2$, $S3$) está operativo (op) o no operativo (no_op).
2. Para cada día dentro del rango de días de la réplica, se determina el estado del proveedor (op o no_op) usando su historial de días sin stock (historial_dias_sin_stock_provee). Mediante un operador ternario en Python que tiene la siguiente estructura (*valor_si_condicion_verdadera* if **condicion** else *valor_si_condicion_falsa)*, se asigna 0 o 1 a las variables S1_op, S2_op, y S3_op. Cuando S1.historial_dias_sin_stock_provee[i] es False, significa que **hay stock**, por lo que S1_op toma el valor 0 para indicar que está **operativo**. Si es True, **no hay stock** y S1_op

toma el valor 1 para indicar que está **no operativo**. Por ejemplo, si la lista S1.historial_dias_sin_stock_provee es: [False, True, False], cuando se itera por los días:

- **Día 1 (i=0):** S1.historial_dias_sin_stock_provee[0] es False. Luego not False es True, por lo tanto, S1_op = 0. (Operativo)
- **Día 2 (i=1):** S1.historial_dias_sin_stock_provee[1] es True. Luego not True es False, por lo tanto, S1_op = 1. (No operativo)
- **Día 3 (i=2):** S1.historial_dias_sin_stock_provee[2] es False. Luego not False es True, por lo tanto, S1_op = 0. (Operativo)

Posteriormente, según el estado (S1_op, S2_op, y S3_op), se incrementará el contador correspondiente (total_S1_op, total_S1_no_op, etc.).

3. Al finalizar la iteración, y contados los días operativos y no operativos (total_S1_op, total_S1_no_op, etc.), se calcula para cada proveedor las probabilidades de que esté (op o no op), dividiendo el número de días en cada estado entre el total de días de la réplica (num_dias):

$$probabilidad\_operativo = \frac{Cantidad\ de\ dias\ que\ el\ proveedor\ estuvo\ operativo}{número\ de\ días\ de\ la\ réplica}$$

$$probabilidad\_no\_operativo = \frac{Cantidad\ de\ dias\ que\ el\ proveedor\ estuvo\ no\ operativo}{número\ de\ días\ de\ la\ réplica}$$

Ejemplo de cálculo de probabilidades para el proveedor S1:

1. probabilidad_operativo_S1 = total_S1_op / num_dias
2. probabilidad_no_operativo_S1 = total_S1_no_op / num_dias

4. Se crea la matriz de probabilidades de cada réplica para cada proveedor (Ejm: *matriz_probabilidades_S1 = [probabilidad_operativo_S1, probabilidad_no_operativo_S1]* y se agrega a la lista *matrices_probabilidades_replicas_S1*, que almacena los resultados de todas las réplicas)

En el caso de los productores $P1$ y $P2$

Las probabilidades condicionales para $P1$ y $P2$ se basan únicamente en sus padres inmediatos en la red. Es decir, se calculan las probabilidades condicionales de que cada productor específico esté en estado operativo o no operativo, en función de las combinaciones de estados de los proveedores.

Como cada productor depende de los tres proveedores ($S1$, $S2$ y $S3$), y tanto los productores como los proveedores pueden estar en los estados "operativo" o "no operativo", se generan múltiples combinaciones. Cada proveedor tiene 8 combinaciones posibles de estados ($2^3 = 8$), y cada productor puede responder de manera diferente a cada una de ellas, por lo tanto, para cada combinación de estados de los proveedores, se determina la probabilidad de que el productor en análisis esté en estado operativo (0) o no operativo (1). Por ejemplo, para la combinación ($S1 = operativo,\ S2 = operativo,\ S3 = no\ operativo$), se calcula tanto la probabilidad de que el productor $P1$ esté operativo (0) como la probabilidad de que esté no operativo (1). Esto da lugar a $8 \times 2 = 16$ probabilidades condicionales a calcular.

Ejemplo de cálculo de probabilidades condicionales para el productor $P1$:

El cálculo implica:

1. Establecimiento de contadores que se utilizarán para registrar el número de ocurrencias de cada combinación de estados de los proveedores ($S1$, $S2$, $S3$), y la combinación conjunta tanto de estados de los proveedores como de $P1$. Para ello se definen dos grupos de contadores:

- **Grupo 1 (sin $P1$)**: Registra la frecuencia de combinaciones de los estados de los proveedores
- **Grupo 2 (con $P1$)**: Registra cuántas veces $P1$ se encuentra "operativo" para cada combinación de estados de los proveedores.

  En total, son 16 contadores, los cuales se presentan en la Tabla 6 junto con su inicialización.

Tabla 6.

Inicialización de contadores para el cálculo de probabilidades condicionales de P1

Grupo 1 (sin $P1$)	Grupo 2 (con $P1$)
total_S1_op_S2_op_S3_op = 0	total_S1_op_S2_op_S3_op_P1_op = 0
total_S1_op_S2_op_S3_no_op = 0	total_S1_op_S2_op_S3_no_op_P1_op = 0
total_S1_op_S2_no_op_S3_op = 0	total_S1_op_S2_no_op_S3_op_P1_op = 0
total_S1_op_S2_no_op_S3_no_op= 0	total_S1_op_S2_no_op_S3_no_op_P1_op = 0
total_S1_no_op_S2_op_S3_op = 0	total_S1_no_op_S2_op_S3_op_P1_op = 0
total_S1_no_op_S2_op_S3_no_op = 0	total_S1_no_op_S2_op_S3_no_op_P1_op = 0
total_S1_no_op_S2_no_op_S3_op = 0	total_S1_no_op_S2_no_op_S3_op_P1_op = 0
total_S1_no_op_S2_no_op_S3_no_op = 0	total_S1_no_op_S2_no_op_S3_no_op_P1_op = 0

2. Para cada día de simulación, los contadores se actualizan reflejando el número de veces que se ha observado la combinación. Se definen las variables (S1_op, S2_op, S3_op, P1_op), a las cuales se les asigna el valor binario de 0 o 1, dependiendo del historial_dias_sin_stock del participante, tal como se explicó anteriormente en el caso de los proveedores. Después de asignar los valores a las variables, el contador correspondiente se incrementa según la combinación evaluada de estados de los proveedores $S1$, $S2$ y $S3$. Además, si $P1$ está operativo para la combinación respectiva, se actualiza el contador asociado. Por ejemplo, si **todos los proveedores están operativos** (S1_op=0, S2_op=0 y S3_op=0), se incrementa total_S1_op_S2_op_S3_op. Adicionalmente, si $P1$ también está operativo, se incrementa total_S1_op_S2_op_S3_op_**P1_op**. Esto para las 8 combinaciones de los proveedores.
3. Cálculo de probabilidades condicionales: Después de iterar y con los contadores actualizados, se calculan las probabilidades condicionales. La probabilidad condicional de que $P1$ esté operativo, dada una combinación específica de estados de $S1$, $S2$, y $S3$, se obtiene dividiendo las veces que $P1$ está operativo en esa combinación entre las veces que la combinación ocurre.

$$Prob\_operativo = \frac{\#\ dias\ del\ Productor\ operativo\ dada\ combinacion\ de\ proveedores}{Total\ de\ días\ de\ ocurrencia\ de\ la\ combinacion\ de\ S1, S2, y\ S3}$$

Como ejemplo, la probabilidad de que $P1$ esté operativo dado que $S1$, $S2$, y $S3$ están operativos. Se calcula de la siguiente manera:

$$prob\_P1\_op\_dado\_S1\_op\_S2\_op\_S3\_op\ = \frac{total\_S1\_op\_S2\_op\_S3\_op\_P1\_op}{total\_S1\_op\_S2\_op\_S3\_op}$$

Como en el grupo 2 solo se registran las combinaciones de los proveedores en las que $P1$ está operativo, es necesario calcular también las probabilidades de que $P1$ esté no operativo para las mismas combinaciones de estados de $S1$, S2, y S3. Para ello, se obtienen las **probabilidades complementarias**, restando de 1 la probabilidad calculada en el paso anterior, lo que permite determinar la probabilidad de que P1 no esté operativo, dada una combinación específica de estados de S1, S2, y S3. Por ejemplo, la probabilidad de que P1 no esté operativo, dado que S1, S2, y S3 están operativos, se calcula de la siguiente manera:

$$prob\_P1\_no\_op\_dado\_S1\_op\_S2\_op\_S3\_op = 1 - prob\_P1\_op\_dado\_S1\_op\_S2\_op\_S3\_op$$

4. Una vez que se han calculado todas las probabilidades condicionales, se construye la matriz de probabilidades (*matriz_probabilidades_P1*) o tabla de probabilidades condicionales (CPT, Conditional Probability Table) que almacena los resultados de la réplica (ver Figura 8), posteriormente, se agrega a la lista *matrices_probabilidades_replicas_P1*, que almacena los resultados de todas las réplicas.

Figura 8.

Ejemplo de configuración de la matriz_probabilidades_P1

```
    # Crear la matriz de probabilidades de P1 para la réplica
    matriz_probabilidades_P1 = np.array([
        [
            prob_P1_op_dado_S1_op_S2_op_S3_op,
            prob_P1_op_dado_S1_op_S2_op_S3_no_op,
            prob_P1_op_dado_S1_op_S2_no_op_S3_op,
            prob_P1_op_dado_S1_op_S2_no_op_S3_no_op,
            prob_P1_op_dado_S1_no_op_S2_op_S3_op,
            prob_P1_op_dado_S1_no_op_S2_op_S3_no_op,
            prob_P1_op_dado_S1_no_op_S2_no_op_S3_op,
            prob_P1_op_dado_S1_no_op_S2_no_op_S3_no_op
        ],
        [
            prob_P1_no_op_dado_S1_op_S2_op_S3_op,
            prob_P1_no_op_dado_S1_op_S2_op_S3_no_op,
            prob_P1_no_op_dado_S1_op_S2_no_op_S3_op,
            prob_P1_no_op_dado_S1_op_S2_no_op_S3_no_op,
            prob_P1_no_op_dado_S1_no_op_S2_op_S3_op,
            prob_P1_no_op_dado_S1_no_op_S2_op_S3_no_op,
            prob_P1_no_op_dado_S1_no_op_S2_no_op_S3_op,
            prob_P1_no_op_dado_S1_no_op_S2_no_op_S3_no_op
        ]
    ])
```

Todos los pasos anteriores se repiten para el caso del productor 2, y en el caso de los demás participantes, se aplica un procedimiento similar.

7.2.2 Probabilidades condicionales como comportamiento medio del participante

Cada replica de la simulación representa el funcionamiento de la cadena durante 1800 días (5 años). Se realizan 5 réplicas, lo que equivale a simular un total de 25 años. Aunque se utilizan los mismos parámetros y condiciones iniciales en cada réplica, los eventos aleatorios pueden diferir generando variaciones en los resultados. Esto implica que, en algunas réplicas, ciertos participantes pueden estar más o menos operativos. Para mitigar este efecto, se calcula el promedio de las probabilidades marginales y condicionales de cada participante de la cadena usando las 5 réplicas, lo que proporciona una estimación más estable del comportamiento del participante. Una vez obtenidos los valores promedio de las réplicas, se crea e imprime la tabla resumen.

En el siguiente ejemplo, se tienen tres réplicas de probabilidades condicionales de $P1$

```
matrices_probabilidades_replicas_P1 = [
    [
        [0.1, 0.2, 0.15, 0.3, 0.05, 0.1, 0.25, 0.2],
        [0.9, 0.8, 0.85, 0.7, 0.95, 0.9, 0.75, 0.8]
    ],
    [
        [0.2, 0.1, 0.25, 0.2, 0.1, 0.3, 0.2, 0.15],
        [0.8, 0.9, 0.75, 0.8, 0.9, 0.7, 0.8, 0.85]
    ],
    [
        [0.15, 0.3, 0.2, 0.25, 0.1, 0.15, 0.3, 0.25],
        [0.85, 0.7, 0.8, 0.75, 0.9, 0.85, 0.7, 0.75]
    ]
]
```

Replica 1

Replica 2

Replica 3

El valor promedio de las probabilidades de las réplicas para este ejemplo es:

```
promedio_probabilidades_P1 =
[
    [0.15, 0.2, 0.2, 0.25, 0.083, 0.183, 0.25, 0.2],  # Promedio de P1 Operativo
    [0.85, 0.8, 0.8, 0.75, 0.917, 0.833, 0.75, 0.8]   # Promedio de P1 No Operativo
]
```

Al aplicar este procedimiento para todos los participantes se obtienen las tablas de probabilidades condicionales (CPTs) que serán utilizadas como parámetros de la BN.

8. Resultados

Después de simular el funcionamiento de 1800 días de la cadena de suministro (réplica), obtener las probabilidades condicionales de la réplica, correr 5 réplicas (con un tiempo de ejecución de aproximadamente 13 minutos), y calcular las probabilidades condicionales promedio, se obtuvieron las siguientes CPTs. Las filas de las tablas corresponden a los estados (Operativo (0) y No Operativo (1)), y las columnas a las diferentes combinaciones de estado de sus nodos padres. La primera fila muestra las probabilidades promedio de que el participante esté operativo para cada combinación de estados de los padres, y la segunda fila las probabilidades promedio de que no esté operativo, también para cada combinación de estados de los padres.

- *Probabilidades de $S1$, $S2$ y $S3$:* Las probabilidades de las variables $S1$, $S2$ y S3 se definen por su distribución individual sin referencia a otras variables ya que no tienen padres.

Probabilidades de $S1$: $S1 = \begin{bmatrix} 0.984333 \\ 0.015667 \end{bmatrix} \quad \begin{matrix} P(S1=0) \\ P(S1=1) \end{matrix}$

Probabilidades de $S2$: $S2 = \begin{bmatrix} 0.988778 \\ 0.011222 \end{bmatrix} \quad \begin{matrix} P(S2=0) \\ P(S2=1) \end{matrix}$

Probabilidades de $S3$: $S3 = \begin{bmatrix} 0.992 \\ 0.008 \end{bmatrix} \quad \begin{matrix} P(S3=0) \\ P(S3=1) \end{matrix}$

- *Probabilidades condicionales de $P1$, $P2$, $W1$, $W2$, $R1$, $R2$ y $R3$ en función de los padres inmediatos de cada variable:* La CPT describe la probabilidad de la variable en función de cada combinación posible de los estados de sus padres.

Tabla de Probabilidades Condicionales de $P1$

$$P1 = \begin{bmatrix} (0,0,0) & (0,0,1) & (0,1,0) & (0,1,1) & (1,0,0) & (1,0,1) & (1,1,0) & (1,1,1) \\ 0.01546 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.98454 & 1.0 & 1.0 & 1.0 & 1.0 & 1.0 & 1.0 & 1.0 \end{bmatrix}$$

Tabla de Probabilidades Condicionales de $P2$

$$P2 = \begin{bmatrix} (0,0,0) & (0,0,1) & (0,1,0) & (0,1,1) & (1,0,0) & (1,0,1) & (1,1,0) & (1,1,1) \\ 0.2585 & 0.0 & 0.075649 & 0.0 & 0.043258 & 0.1 & 0.1 & 0.0 \\ 0.7415 & 1.0 & 0.924351 & 1.0 & 0.956742 & 0.9 & 0.9 & 1.0 \end{bmatrix}$$

Tabla de Probabilidades Condicionales de $W1$

$$W1 = \begin{bmatrix} (0,0) & (0,1) & (1,0) & (1,1) \\ 0.969424 & 0.726573 & 0.754836 & 0.663411 \\ 0.030576 & 0.273427 & 0.245164 & 0.336589 \end{bmatrix}$$

Tabla de Probabilidades Condicionales de $W2$

$$W2 = \begin{bmatrix} (0,0) & (0,1) & (1,0) & (1,1) \\ 0.860652 & 0.578601 & 0.546523 & 0.399211 \\ 0.139348 & 0.421399 & 0.453477 & 0.600789 \end{bmatrix}$$

Tabla de Probabilidades Condicionales de $R1$

$$R1 = \begin{bmatrix} (0) & (1) \\ 0.978231 & 0.994352 \\ 0.021769 & 0.005648 \end{bmatrix}$$

Tabla de Probabilidades Condicionales de $R2$

$$R2 = \begin{bmatrix} (0) & (1) \\ 0.954937 & 0.973015 \\ 0.045063 & 0.026985 \end{bmatrix}$$

Tabla de Probabilidades Condicionales de $R3$

$$R3 = \begin{bmatrix} (0) & (1) \\ 0.913718 & 0.955815 \\ 0.086282 & 0.044185 \end{bmatrix}$$

8.1 Creación de la Red Bayesiana (BN)

El núcleo de la representación de la BN es el gráfico dirigido acíclico, cuyos nodos son las variables aleatorias en análisis, y las aristas entre los nodos la influencia directa de un nodo sobre otro (Koller & Friedman, 2009). Como se mencionó, la estructura del modelo de BN se construirá en base a la dinámica del funcionamiento de la cadena, es decir, las relaciones entre cada uno de los participantes. Para crear la red, se siguen los siguientes pasos:

Paso 1: Identificación de variables

En el dominio del sistema en análisis, las variables son: $S1, S2, S3, P1, P2, W1, W2, R1, R2,$ y $R3$.

Paso 2: Definición de relaciones entre las variables:

- Relaciones iniciales: $S1, S2,$ y $S3$ son nodos padres de $P1$ y $P2$.
- Relaciones intermedias: $P1$ es un nodo padre de $W1$ y $W2$, al igual que $P2$, que también es nodo padre de $W1$ y $W2$.

- Relaciones finales: $W1$ es un nodo padre de $R1$, mientras que $W2$ es un nodo padre de $R2$ y $R3$.

Paso 3: Construcción del diagrama de red bayesiana

- Dibujar los nodos: Las variables aleatorias de la BN se representan por nodos, cada una visualizada como un círculo en el grafo. Se dibujan nodos para $S1, S2, S3, P1, P2, W1, W2, R1, R2$, y $R3$. Los estados de cada nodo en la BN son binarios, con dos posibles estados: "Operativo" (0) o "No Operativo" (1).
- Dibujar las aristas: Se dibujan aristas **(**arcos dirigidos que representan dependencias condicionales) entre los nodos según la estructura especificada en el paso 2, donde se definieron las relaciones entre las variables.

Finalmente, se asignan las CPTs que fueron previamente calculadas a partir de los resultados de la simulación de la dinámica de la cadena de suministro. El modelo de BN resultante que ilustra las relaciones causales y probabilísticas entre el conjunto de variables, se presenta en la Figura 9.

Figura 9.

Modelo de Red Bayesiana

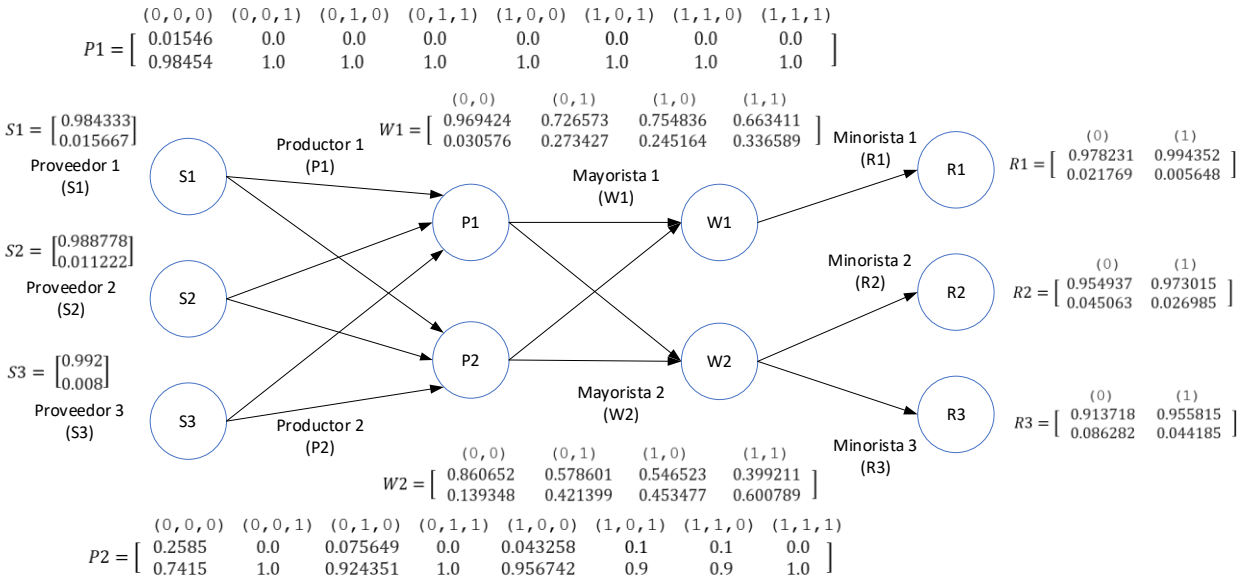


Figura 10.

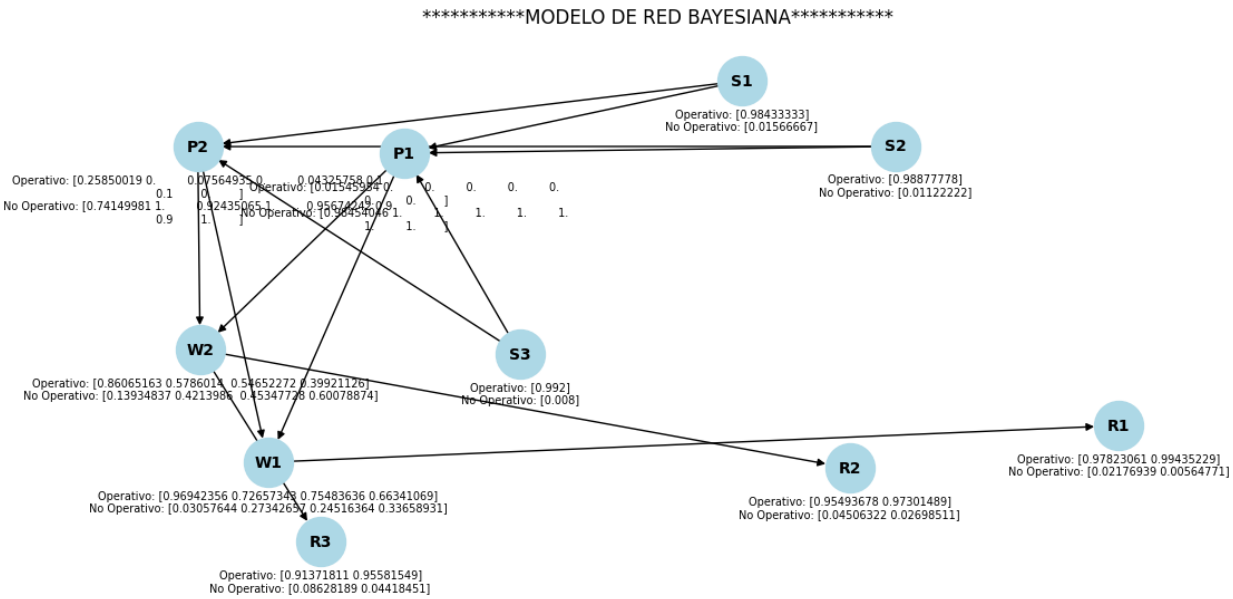
Pseudocódigo del modelo de red bayesiana

```
// Instalar la biblioteca pgmpy si no está instalada
// Importar las clases necesarias desde la biblioteca pgmpy:
      - BayesianNetwork para definir la estructura de la red bayesiana
      - TabularCPD para definir las Tablas de Probabilidad Condicional (CPT)
      para los nodos de la red bayesiana
      - VariableElimination para realizar inferencia sobre la red bayesiana

// Definir el modelo de red bayesiana con relaciones entre nodos (arcos
dirigidos): Se crea un objeto llamado modelo con una lista de tuplas que
representan las relaciones (arcos dirigidos) entre los nodos (variables) en
la red. Entiéndase ('S1', 'P1') como que S1 es padre de P1, lo que
significa que P1 depende de S1.
modelo = BayesianNetwork([('S1', 'P1'), ('S2', 'P1'), ('S3', 'P1'),
                          ('S1', 'P2'), ('S2', 'P2'), ('S3', 'P2'),
                          ('P1', 'W1'), ('P1', 'W2'),
                          ('P2', 'W1'), ('P2', 'W2'),
                          ('W1', 'R1'),
                          ('W2', 'R2'), ('W2', 'R3')])
// Definir las Tablas de Probabilidad Condicional (CPT) para cada nodo.
Cada CPT define cómo las probabilidades de un nodo dependen de sus padres:
- Para cada nodo (S1, S2, S3, P1, P2, W1, W2, R1, R2, R3):
      - Se crea la tabla utilizando TabularCPD con:
        - El nombre del nodo para el cual se está definiendo la CPT (variable).
        - La cardinalidad de la variable variable_card o el número de estados
        posibles (en este caso, 2: operativo y no operativo).
        - Las probabilidades para cada combinación de los estados de los
        padres (values).
        - Si aplica, especificar los padres de la variable y sus
        cardinalidades (evidence y evidence_card).
      Por ejemplo:
      # Definir la CPT para S1 con las probabilidades promedio
      S1_cpd = TabularCPD(variable='S1', variable_card=2,
                          values=[[0.984333], [0.015667]])
      En este caso se define una CPT para el nodo S1 con dos estados (0 y
      1) y sus probabilidades correspondientes (0.984333 para el estado 0 y
      0.015667 para el estado 1).

// Asignar las CPT a los nodos en el modelo de red bayesiana (Se añaden
todas las CPT definidas a sus respectivos nodos en el modelo):
modelo.add_cpds(S1_cpd, S2_cpd, S3_cpd, P1_cpd, P2_cpd, W1_cpd, W2_cpd,
R1 cpd, R2 cpd, R3 cpd)
```

Este modelo se implementó en Python utilizando la biblioteca *pgmpy*. La estructura de la red, que representa las relaciones causales y de dependencia entre los nodos, se definió mediante la clase *BayesianNetwork* de dicha biblioteca. De manera similar, la CPT de cada nodo (probabilidades de los diferentes estados del nodo, dados los estados de sus nodos padres) se establecieron usando la clase *TabularCPD*. Finalmente, las CPTs introducidas se asignaron a los nodos correspondientes de la red bayesiana. El pseudocódigo de implementación se presenta en la Figura 10, y la visualización en Python del modelo implementado es la que se observa en la Figura 11.

Figura 11.

Visualización en Python del modelo de red bayesiana



8.2 Análisis de riesgos

Una vez construida la red bayesiana, se procede a predecir el riesgo de interrupción en un nodo específico de la cadena de suministro. Este proceso implica calcular la probabilidad marginal de interrupción para cada nodo, lo que requiere marginalizar (sumar) sobre todas las variables que influyen en el nodo en cuestión.

Cálculo de Probabilidades Marginales:

- *Probabilidades marginales de las variables sin padres* ($S1, S2, y\ S3$):

Para las variables que no tienen padres, la probabilidad marginal se obtiene directamente de las tablas de probabilidad ingresadas para representar la red bayesiana, las cuales reflejan la distribución de probabilidad de cada variable de forma independiente ya que no dependen de ninguna otra. Las probabilidades marginales de *S1, S2, y* $S3$ son:

$$S1 = \begin{bmatrix} 0.984333 \\ 0.015667 \end{bmatrix} \qquad S2 = \begin{bmatrix} 0.988778 \\ 0.011222 \end{bmatrix} \qquad S3 = \begin{bmatrix} 0.992 \\ 0.008 \end{bmatrix}$$

- *Probabilidades marginales de las variables con padres P1, P2, W1, W2, R1, R2, R3:*

Para obtener la probabilidad marginal en el caso de las variables con padres, se determina todas las configuraciones posibles de las variables parentales, se multiplica las probabilidades de las variables según sus tablas condicionales y marginales, y se suman estas probabilidades.

Fórmulas para el cálculo de probabilidades marginales totales de cada variable

Para el caso en análisis, la probabilidad marginal $P(P1 = p1)$ es la probabilidad marginal de que la variable $P1$ tome un valor específico $p1$ en la red bayesiana, independientemente de los valores específicos que tomen $S1$, *S2*, y *S3*.

Para ello, se suma sobre todas las combinaciones posibles de los valores que pueden tomar $S1$, $S2$, y $S3$ así:

$$P(P1 = p1) = \sum_{S1}\sum_{S2}\sum_{S3}[P(P1 = p1 \mid S1, S2, S3) \cdot P(S1) \cdot P(S2) \cdot P(S3)] \qquad \textbf{(2)}$$

Donde:

- $P(P1 = p1 \mid S1, S2, S3)$: Es la probabilidad condicional de que $P1$ tome el valor de $p1$ dado que $S1, S2, S3$ toman valores específicos

- $P(S1)$, $P(S2)$, $P(S3)$: Son las probabilidades marginales de $S1$, $S2$, y $S3$ o las probabilidades de que $S1$, $S2$, y $S3$ tomen ciertos valores, independientemente de los otros nodos. En este caso son las mismas que las dadas porque no dependen de ninguna otra variable en la red.

El mismo procedimiento se realiza para $P2$

$$P(P2 = p2) = \sum_{S1}\sum_{S2}\sum_{S3} P(P2 = p2 \mid S1, S2, S3) \cdot P(S1) \cdot P(S2) \cdot P(S3) \quad \textbf{(3)}$$

En el caso de $W1$ la probabilidad marginal sería:

$$P(W1 = w1) = \sum_{S1}\sum_{S2}\sum_{S3}\sum_{P1}\sum_{P2} P(W1 = w1 \mid P1, P2) \cdot P(P1 \mid S1, S2, S3) \cdot P(P2 \mid S1, S2, S3) \cdot P(S1) \cdot P(S2) \cdot P(S3) \quad \textbf{(4)}$$

El mismo procedimiento de $W1$ se realiza para $W2$ dado que las mismas variables influyen en él.

$$P(W2 = w2) = \sum_{S1}\sum_{S2}\sum_{S3}\sum_{P1}\sum_{P2} P(W2 = w2 \mid P1, P2) \cdot P(P1 \mid S1, S2, S3) \cdot P(P2 \mid S1, S2, S3) \cdot P(S1) \cdot P(S2) \cdot P(S3) \quad \textbf{(5)}$$

En el caso de R1, R2 y R3 la probabilidad marginal es:

$$P(R1 = r1) = \sum_{S1}\sum_{S2}\sum_{S3}\sum_{P1}\sum_{P2}\sum_{W1} P(R1 = r1 \mid W1) \cdot P(W1 \mid P1, P2) \cdot P(P1 \mid S1, S2, S3) \cdot P(P2 \mid S1, S2, S3) \cdot P(S1) \cdot P(S2) \cdot P(S3) \quad \textbf{(6)}$$

$$P(R2 = r2) = \sum_{S1}\sum_{S2}\sum_{S3}\sum_{P1}\sum_{P2}\sum_{W2} P(R2 = r2 \mid W2) \cdot P(W2 \mid P1, P2) \cdot P(P1 \mid S1, S2, S3) \cdot P(P2 \mid S1, S2, S3) \cdot P(S1) \cdot P(S2) \cdot P(S3) \quad \textbf{(7)}$$

$$P(R3 = r3) = \sum_{S1}\sum_{S2}\sum_{S3}\sum_{P1}\sum_{P2}\sum_{W2} P(R3 = r3 \mid W2) \cdot P(W2 \mid P1, P2) \cdot P(P1 \mid S1, S2, S3) \cdot P(P2 \mid S1, S2, S3) \cdot P(S1) \cdot P(S2) \cdot P(S3) \quad \textbf{(8)}$$

El cálculo de las probabilidades marginales totales puede simplificarse utilizando el algoritmo de eliminación de variables**.** En este sentido, la biblioteca *pgmpy* ofrece una implementación del algoritmo mencionado, lo que permite calcular las probabilidades marginales de forma eficiente.

Figura 12.

Resultados de las probabilidades marginales usando pgmpy

```
Sin evidencia:
P(S1=0) (Operativo) = 0.9843333333333334
P(S1=1) (No operativo) = 0.01566666666666667
P(S2=0) (Operativo) = 0.9887777777777778
P(S2=1) (No operativo) = 0.011222222222222222
P(S3=0) (Operativo) = 0.992
P(S3=1) (No operativo) = 0.008
P(P1=0) (Operativo) = 0.014926190935407374
P(P1=1) (No operativo) = 0.9850738090645926
P(P2=0) (Operativo) = 0.25110563540701175
P(P2=1) (No operativo) = 0.7488943645929883
P(W1=0) (Operativo) = 0.6878952266436735
P(W1=1) (No operativo) = 0.31210477335632647
P(W2=0) (Operativo) = 0.43939949090380065
P(W2=1) (No operativo) = 0.5606005090961993
P(R1=0) (Operativo) = 0.9832622654937235
P(R1=1) (No operativo) = 0.016737734506276518
P(R2=0) (Operativo) = 0.965071374758719
P(R2=1) (No operativo) = 0.034928625241281014
P(R3=0) (Operativo) = 0.9373179228430811
P(R3=1) (No operativo) = 0.0626820771569189
************************************************
```

En la Figura 12 se presentan los resultados de las probabilidades marginales para cada una de las variables involucradas en el modelo, calculadas mediante el método de eliminación de variables. Esto representa la probabilidad de que los diferentes actores de la cadena estén en estado operativo (0) o no operativo (1), sin la presencia de evidencia adicional.

Los resultados indican que, los tres proveedores ($S1$, $S2$, y $S3$) tienen una alta probabilidad de estar operativos (98.43%, 98.88% y 99.2%), lo que sugiere que, en el contexto del modelo actual, son bastante confiables y es poco probable que una interrupción se origine de su parte. En contraste, el productor 1 ($P1$) presenta una probabilidad alta de no estar operativo (98.51%), lo que lo convierte en un punto crítico de la cadena. El productor 2 ($P2$), por su parte, muestra una probabilidad algo menor de no estar operativo (74.89%). Estas probabilidades convierten a ambos productores en cuellos de botella que limitan el flujo de productos desde los proveedores hacia los mayoristas. Sin embargo, dado que ambos suministran el mismo producto, la inoperatividad de uno puede ser compensada por la operatividad del otro.

En el caso de los mayoristas ($W1$) y ($W2$), tienen una probabilidad menor de estar no operativos (31.21% y 56.06%). Esto indica que ambos mayoristas también son puntos vulnerables en la cadena, con altas posibilidades de operar con interrupciones.

En cuanto a los minoristas ($R1$, $R2$, $R3$), nodos de gran importancia para analizar el riesgo de interrupción de la SC, medido en términos del eslabón más cercano al consumidor o último participante aguas abajo, se observa una alta probabilidad de estar operativos (98.33%, 96.51% y 93.73%), siendo $R3$ quien presenta mayor probabilidad de estar no operativo (6.27%), reflejando una capacidad ligeramente limitada para garantizar operaciones ininterrumpidas.

Se realizó la validación de los resultados, al compararlos con los obtenidos mediante el desarrollo de las ecuaciones (2) a (8), cuyos resultados se presentan en la Tabla 7, en la columna titulada 'Fórmula teórica'. Esta comparación confirmó la precisión del algoritmo de eliminación de variables implementado en *pgmpy*, asegurando que los resultados son consistentes y fiables, dado que el porcentaje de error es nulo para todas las variables (ver Tabla 7).

Tabla 7.

Resultados de las probabilidades marginales y cálculo de % de error

Probabilidad marginal	pgmpy.models	Fórmula teórica	% de Error
P(S1=0)	0.9843333333333334	0.984333	0.00%
P(S1=1)	0.01566666666666667	0.015667	0.00%
P(S2=0)	0.9887777777777778	0.988778	0.00%
P(S2=1)	0.011222222222222222	0.011222	0.00%
P(S3=0)	0.992	0.992	0.00%
P(S3=1)	0.008	0.008	0.00%
P(P1=0)	0.014926190935407374	0.01492664	0.00%
P(P1=1)	0.9850738090645926	0.98507336	0.00%
P(P2=0)	0.25110563540701175	0.25110542	0.00%
P(P2=1)	0.7488943645929883	0.74889458	0.00%
P(W1=0)	0.6878952266436735	0.68789539	0.00%
P(W1=1)	0.31210477335632647	0.31210461	0.00%
P(W2=0)	0.43939949090380065	0.43939943	0.00%
P(W2=1)	0.5606005090961993	0.56060057	0.00%
P(R1=0)	0.9832622654937235	0.98326244	0.00%
P(R1=1)	0.016737734506276518	0.01673756	0.00%
P(R2=0)	0.965071374758719	0.96507154	0.00%
P(R2=1)	0.034928625241281014	0.03492846	0.00%
P(R3=0)	0.9373179228430811	0.9373176	0.00%
P(R3=1)	0.0626820771569189	0.0626824	0.00%

8.3 Análisis de resiliencia

Se propone un enfoque basado en la estructura probabilística del modelo y la propagación de evidencia dentro de la red bayesiana, utilizando la variación en la probabilidad de los nodos objetivo como medida de resiliencia. Este enfoque evalúa la capacidad de los nodos para mantener su estado operativo frente a interrupciones, reflejando cómo el sistema reacciona y se adapta a eventos disruptivos. A medida que la evidencia de un evento disruptivo se propaga, la variación en las probabilidades de los nodos objetivo, sirve como indicador del grado en que el sistema puede mantener su funcionamiento en presencia de vulnerabilidades.

La propuesta se fundamenta en el concepto de resiliencia desarrollado por Heckmann, Comes & Nickel (2015), definido como "la capacidad de una cadena de suministro para superar la vulnerabilidad", entendida esta última como el grado en que la cadena de suministro es susceptible a un evento de riesgo específico. Esto implica que la resiliencia está directamente relacionada con la capacidad del sistema para enfrentar los efectos adversos de interrupciones y difiere del planteamiento de Hosseini e Ivanov (2019) al no considerar la recuperabilidad. La propuesta presentada evalúa directamente el impacto de la alteración (interrupción en un actor específico) sobre nodos clave de la cadena (minoristas $R1, R2$, o $R3$). El cálculo de la resiliencia se lleva a cabo mediante los siguientes pasos:

Cálculo de resiliencia basada en cambios probabilísticos:

1. Definir la probabilidad base ($\boldsymbol{B_O}$) o probabilidad de que el nodo objetivo esté en estado operativo (0) bajo condiciones normales (sin modificar ningún otro nodo en la red).

$$\boldsymbol{B_O} = P(x_{objetivo} = operativo)$$

2. Calcular la probabilidad $\boldsymbol{B}_{(\boldsymbol{O}|\bar{X}i)}$ o probabilidad de que el nodo objetivo esté operativo (0) cuando el participante $x_i \neq x_{objetivo}$ está en estado no operativo (1). Esto es fijando evidencia del estado del participante x_i a no operativo

$$\boldsymbol{B}_{(\boldsymbol{o}|\bar{X}i)} = P(x_{objetivo} = operativo \mid x_i = no\ operativo)$$

3. Evaluar la resiliencia, utilizando una métrica que mide la capacidad del nodo objetivo para mantener su estado operativo frente a la variación en el estado del participante x_i. La resiliencia cuando el participante x_i está no operativo (**resiliencia no operativo**) se define como:

$$Resiliencia = 1 + \left|\frac{\boldsymbol{B}_{(\boldsymbol{o}|\bar{X}i)} - \boldsymbol{B}_{\boldsymbol{o}}}{\boldsymbol{B}_{\boldsymbol{o}}}\right|$$

La resiliencia calculada de esta manera no es evaluada de manera aislada; mide cómo varía la probabilidad del nodo objetivo al cambiar el estado de un actor a no operativo, reflejando el comportamiento del nodo en el contexto de toda la red.

- Una resiliencia cercana a 1 indica que el nodo objetivo es estable y no se ve mayormente afectado por la variación en el estado del participante x_i.
- Una resiliencia menor a 1 indica que el nodo objetivo es vulnerable y se ve afectado negativamente por el estado de no operatividad del participante x_i. Esto refleja que el nodo objetivo tiene dependencias importantes en el participante x_i y no logra mantenerse operativo de manera eficiente. Si el nodo depende en gran medida de que un participante esté operativo, su resiliencia será baja frente a la no operación de ese participante.
- Una resiliencia mayor a 1 indica que el nodo objetivo tiene una mayor resiliencia frente a ese participante y su no operatividad, mejorando su desempeño. En este caso, este valor es visto como que el nodo objetivo tiene capacidad de compensar o adaptarse bien ante esta situación.

La implementación del cálculo de resiliencia se realizó en Python. Los resultados para los tres nodos objetivo (minoristas $R1$, $R2$, y $R3$) de la cadena de suministro, en función de su afectación en el estado de operación cuando cambia a no operativo el estado de otro participante se resumen en la Figura 13.

En cuanto al nodo objetivo R1: Los valores de resiliencia de $R1$ cuando la mayoría de los participantes están no operativos se encuentran muy cercanos a 1, lo que sugiere que $R1$ es muy estable incluso cuando los participantes no están operativos. $R1$ es resiliente cuando la mayoría de los participantes están no operativos.

En cuanto al nodo objetivo R2: Los valores de resiliencia para $R2$ son similares a los observados en $R1$. Cuando la mayoría de los participantes están no operativos es cercana a 1, lo que sugiere que $R2$ tiene un comportamiento bastante estable ante la no operatividad de los actores.

En cuanto al nodo objetivo R3: Para $R3$, la resiliencia muestra un comportamiento similar al de $R1$ y $R2$. Los valores están cercanos a 1 indicando que es resiliente a la no operatividad de los actores.

Figura 13.

Resultados de cálculo de resiliencia

```
**********************************************
ANÁLISIS DE RESILIENCIA
**********************************************
Tabla para R1
Participante      B_O  B_O_X_bar_i  Resiliencia No Operativo
          S1 0.983262     0.983592                  1.000335
          S2 0.983262     0.983546                  1.000288
          S3 0.983262     0.983655                  1.000399
          P1 0.983262     0.983287                  1.000025
          P2 0.983262     0.983642                  1.000386
          W1 0.983262     0.994352                  1.011279
          W2 0.983262     0.983349                  1.000088
          R2 0.983262     0.983237                  0.999974
          R3 0.983262     0.983230                  0.999967
Tabla para R2
Participante      B_O  B_O_X_bar_i  Resiliencia No Operativo
          S1 0.965071     0.965680                  1.000630
          S2 0.965071     0.965597                  1.000545
          S3 0.965071     0.965794                  1.000749
          P1 0.965071     0.965129                  1.000060
          P2 0.965071     0.965750                  1.000703
          W1 0.965071     0.965246                  1.000181
          W2 0.965071     0.973015                  1.008231
          R1 0.965071     0.965019                  0.999946
          R3 0.965071     0.962081                  0.996901
Tabla para R3
Participante      B_O  B_O_X_bar_i  Resiliencia No Operativo
          S1 0.937318     0.938735                  1.001512
          S2 0.937318     0.938542                  1.001306
          S3 0.937318     0.939000                  1.001795
          P1 0.937318     0.937453                  1.000144
          P2 0.937318     0.938898                  1.001686
          W1 0.937318     0.937725                  1.000434
          W2 0.937318     0.955815                  1.019735
          R1 0.937318     0.937196                  0.999870
          R2 0.937318     0.931951                  0.994274
**********************************************
```

9. Análisis de Sensibilidad

Un método efectivo para evaluar la validez de un modelo desarrollado por expertos es llevar a cabo un análisis de sensibilidad. Este análisis permite visualizar de manera gráfica cuáles variables ejercen mayor impacto sobre un nodo objetivo (Fenton & Neil, 2013, como se cita en Hosseini & Ivanov, 2021). Lo cual permite verificar si el modelo de BN está funcionando como se espera, de acuerdo a las expectativas que se tiene sobre el comportamiento de las variables.

Considerando lo anterior, se realiza el siguiente análisis de sensibilidad de los minoristas bajo ciertas evidencias. A partir de diferencias entre las probabilidades de que los minoristas estén operativos o no, en función de las evidencias de un participante determinado, se elaboran los gráficos de tornado presentados en las Figura 14 y 15. Por ejemplo, en el caso de que $R3$ esté operativo bajo evidencias relacionadas con el estado de $W2$, la diferencia se calcula como:

$$0.9137181074091784 - 0.9558154923579548 \approx -0.042097$$

- Evidencia: $W2$ Operativo $\rightarrow P(R3 = 0 \mid W2\ Operativo) = 0.9137181074091784$
- Evidencia: $W2$ No Operativo $\rightarrow P(R3 = 0 \mid W2\ No\ Operativo) = 0.9558154923579548$

Figura 14.

Gráfico del Análisis de Sensibilidad - Operativo

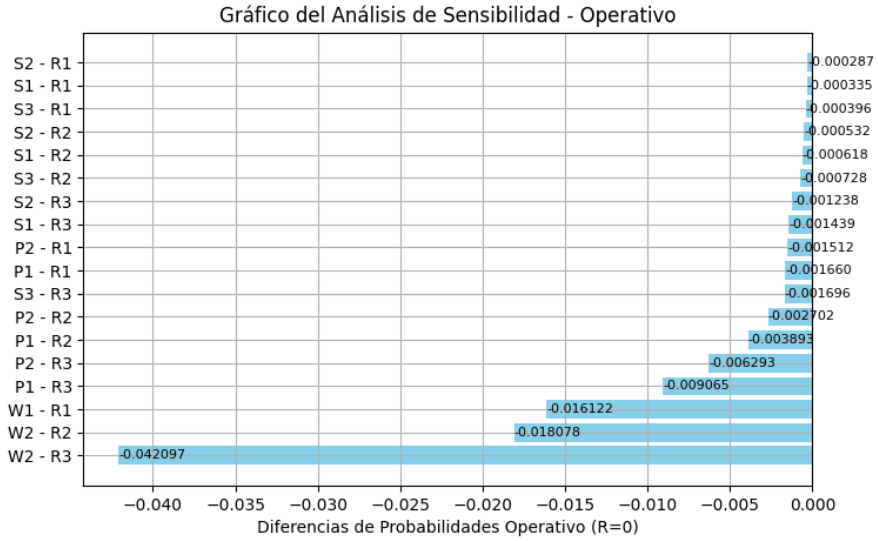
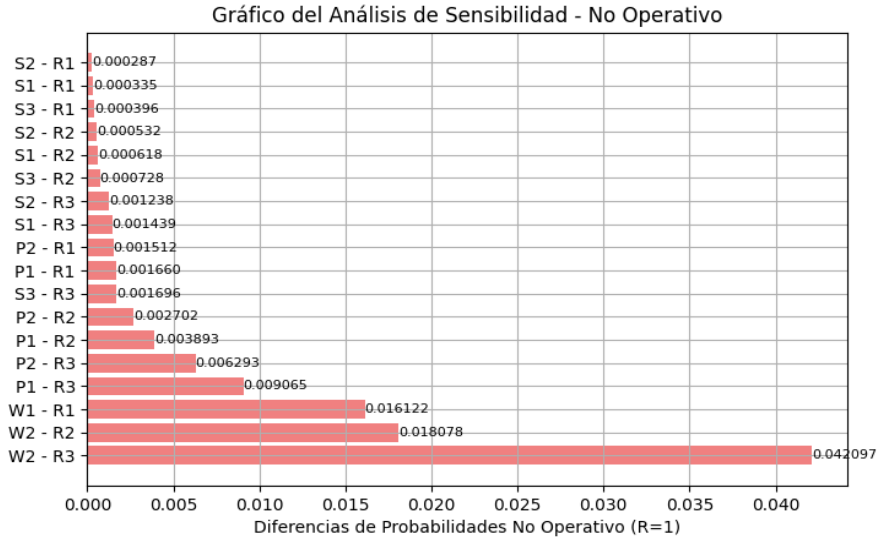


Figura 15.

Gráfico del Análisis de Sensibilidad – No Operativo



La longitud de la barra en las gráficas puede interpretarse como una medida del impacto del participante sobre el minorista en análisis.

La diferencia de aproximadamente -0.042097 es pequeña. A pesar de que $R3$ tiene una probabilidad ligeramente menor de estar operativo cuando $W2$ está operativo, esta diferencia es mínima y limitada en cuanto al impacto de $W2$ sobre $R3$. Esto sugiere que $R3$ mantiene una alta probabilidad de operatividad en ambos casos, tanto cuando $W2$ está en operativo como cuando no lo está.

En general, las diferencias observadas en el análisis de sensibilidad para todas las variables son pequeñas, lo que puede interpretarse como una estabilidad en el comportamiento de los minoristas, independientemente del estado de los demás actores, estando en concordancia con los resultados de sus probabilidades marginales. Este análisis permite validar el comportamiento de la red, sugiriendo que el modelo está correctamente especificado (consistente con las expectativas de funcionamiento de la red hipotética).

10. Conclusiones

El modelo propuesto, basado en redes bayesianas confirma la hipótesis planteada al iniciar la investigación: “El modelo de redes bayesianas a proponer mejora la comprensión de los riesgos y resiliencia en una cadena de suministro”, lo anterior debido a que permite determinar la probabilidad de operación de la cadena bajo condiciones actuales, simular escenarios disruptivos y predecir su comportamiento. Además, facilita la identificación de efectos que no son evidentes a simple vista, respaldando la toma de decisiones estratégicas informadas. Por ejemplo, en la cadena hipotética simulada, de los tres nodos minoristas, $R1$ muestra la mayor probabilidad de estar operativo, mientras que $R3$ el más vulnerable, sería prioritario para mitigación.

La metodología propuesta para la creación del modelo es flexible y puede aplicarse en la práctica a la dinámica de funcionamiento de cualquier cadena de suministro que requiera análisis. Esto mediante el ajuste en Python de la cantidad de actores y su comportamiento según el contexto específico; obteniendo de esta manera los parámetros asociados (probabilidades condicionales), con los cuales se realizan los análisis de riesgos y resiliencia necesarios para la toma de decisiones.

Por otro lado, dado que no se contó con un grupo de expertos para la validación del modelo propuesto al ser esta una cadena hipotética, es claro que los resultados obtenidos están sujetos a los escenarios tratados que podrían en algunos casos diferir de situaciones y aplicaciones concretas de la realidad, como por ejemplo el no incluir los costos asociados a la operación e interrupciones.

El análisis de sensibilidad realizado refleja que el modelo es consistente con lo esperado. Sin embargo, en el evento de una replicación del funcionamiento de una cadena real, disponer de información de expertos en la materia sería ideal para utilizar varios enfoques de validación de manera conjunta, lo que permitirá asegurar que el modelo refleja adecuadamente el comportamiento del sistema real, reforzando su validez.

11. Recomendaciones

Realizar manualmente todas las sumatorias para el cálculo de una probabilidad marginal implica recorrer todas las combinaciones de las variables lo cual no es eficiente, ya que a medida que aumenta el número de éstas, la cantidad de combinaciones posibles crece exponencialmente. Adicionalmente, se calculan varias veces las mismas combinaciones y probabilidades, por lo tanto, un algoritmo como el de eliminación de variables de la biblioteca *pgmpy* evita recalcular estas combinaciones, reutilizando los cálculos donde es posible y haciendo eficiente el proceso.

El modelo conceptual desarrollado puede por ejemplo para futuras investigaciones expandirse a cadenas de suministro más complejas para una visión aún más detallada de los riesgos y resiliencia en otros contextos. Por otro lado, la metodología podría combinarse con técnicas de optimización para la toma de decisiones estratégicas.

Otro campo de investigación a considerar podría ser la creación de modelos híbridos que combinen redes bayesianas con otras técnicas de análisis predictivo, para lograr modelos más robustos y con mayor capacidad de predicción.

Adicionalmente, se podría continuar investigando, usando un enfoque de Red Bayesiana Dinámica (DBN) considerando las variaciones en el tiempo de manera explícita, agregando más nodos a la red para representar cada participante en diferentes momentos del tiempo; por ejemplo, $P1\_t0$, $P1\_t1$, modelando transiciones explícitas desde un estado disruptivo a uno de recuperación.

Una limitación del presente estudio es la complejidad de la simulación y la dependencia de los datos de la simulación para la obtención de las probabilidades condicionales (pero esto es precisamente lo que enriquece el modelo, lo hace riguroso, lo aleja de interpretaciones subjetivas y permite su flexibilidad para involucrar diferentes contextos); en este sentido se debe ser muy preciso en cuanto a la dinámica de funcionamiento de la cadena que se quiera representar.

Referencias Bibliográficas

Axsäter, S. (2006). *Inventory control* (2.a ed.). Springer Science+Business Media.

Babu, H., Bhardwaj, P., & Agrawal, A. K. (2020). Modelling the supply chain risk variables using ISM: A case study on Indian manufacturing SMEs. *Journal of Modelling in Management, 16*(1), 215–239. https://doi.org/10.1108/JM2-06-2019-0126

Badhotiya, G. K., Soni, G., Jain, V., Joshi, R., & Mittal, S. (2022). Assessing supply chain resilience to the outbreak of COVID-19 in Indian manufacturing firms. *Operations Management Research*. https://doi.org/10.1007/s12063-021-00236-6

Badurdeen, F., Shuaib, M., Wijekoon, K., Brown, A., Faulkner, W., Amundson, J., Jawahir, I. S., J. Goldsby, T., Iyengar, D., & Boden, B. (2014). Quantitative modeling and analysis of supply chain risks using Bayesian theory. *Journal of Manufacturing Technology Management*, 25(5), 631–654. https://doi.org/10.1108/jmtm-10-2012-0097

Behzadi, G., O'Sullivan, M. J., & Olsen, T. L. (2020). On metrics for supply chain resilience. *European Journal of Operational Research*, 287(1), 145–158. https://doi.org/10.1016/j.ejor.2020.04.040

Bouzembrak, Y., & Marvins, H. J. P. (2019). Impact of drivers of change, including climatic factors, on the occurrence of chemical food safety hazards in fruits and vegetables: A Bayesian Network approach. *Food Control, 97*, 67–76. https://doi.org/10.1016/j.foodcont.2018.10.021

Brandon-Jones, E., Squire, B., Autry, C., & Petersen, K. J. (2014). A contingent resource-based perspective of supply chain resilience and robustness. *Journal of Supply Chain Management*, 50(3), 55–73. https://doi.org/10.1111/jscm.12050

Bugert, N., & Lasch, R. (2023). Analyzing upstream and downstream risk propagation in supply networks by combining agent-based modeling and Bayesian networks. *Journal of Business Economics*, 93, 859–889 (2023). https://doi.org/10.1007/s11573-022-01128-2.

Chhimwal, M., Agrawal, S., & Kumar, G. (2021). Measuring circular supply chain risk: A Bayesian network methodology. *Sustainability*, 13(15), 8448. https://doi.org/10.3390/su13158448

Chopra, S., & Meindl, P. (2008). *Administración de la cadena de suministro: Estrategia, planeación y operación* (3ra ed.). Pearson Educación.

Çıkmak, S., & Ungan, M. C. (2022). Supply chain risks and mitigation strategies in Turkey automotive industry: findings from a mixed-method approach. *Supply Chain Forum: An International Journal*, 1–21. https://doi.org/10.1080/16258312.2022.2060694

Dong, Q., & Cooper, O. (2016). An orders-of-magnitude AHP supply chain risk assessment framework. *International Journal of Production Economics, 182*, 144–156. https://doi.org/10.1016/j.ijpe.2016.08.021

Fan, S., Yang, Z., Wang, J., & Marsland, J. (2022). Shipping accident analysis in restricted waters: Lesson from the Suez Canal blockage in 2021. *Ocean Engineering*, 266, 113119. https://doi.org/10.1016/j.oceaneng.2022.113119

Fang, Y.-P., & Zio, E. (2019). An adaptive robust framework for the optimization of the resilience of interdependent infrastructures under natural hazards. *European Journal of Operational Research*, *276*(4), 1119–1136. https://doi.org/10.1016/j.ejor.2019.01.052

Fattahi, M., Govindan, K., & Keyvanshokooh, E. (2017). Responsive and resilient supply chain network design under operational and disruption risks with delivery lead-time sensitive

customers. *Transportation Research Part E: Logistics and Transportation Review, 101*, 176-200. https://doi.org/10.1016/j.tre.2017.02.004

Gui, D., Wang, H., & Yu, M. (2022). Risk assessment of port congestion risk during the COVID-19 Pandemic. *Journal of Marine Science and Engineering*, 10(2), 150. https://doi.org/10.3390/jmse10020150

Heckmann, I., Comes, T., & Nickel, S. (2015). A critical review on supply chain risk – Definition, measure and modeling. *Omega*, 52, 119–132. https://doi.org/10.1016/j.omega.2014.10.004

Heidari, S. S., Khanbabaei, M., & Sabzehparvar, M. (2018). A model for supply chain risk management in the automotive industry using fuzzy analytic hierarchy process and fuzzy TOPSIS. *Benchmarking: An International Journal*, *25*(9), 3831–3857.

Heizer, J., & Render, B. (2009). *Principios de administración de operaciones* (7.ª ed.). Pearson Educación.

Henry, D., & Ramirez-Marquez, J. E. (2012). Generic metrics and quantitative approaches for system resilience as a function of time. *Reliability Engineering & System Safety*, *99*(1), 114–122. https://doi.org/10.1016/j.ress.2011.09.002

Hosseini, S., Al Khaled, A., & Sarder, M. (2016). A general framework for assessing system resilience using Bayesian networks: A case study of sulfuric acid manufacturer. *Journal of Manufacturing Systems*, 41, 211–227. https://doi.org/10.1016/j.jmsy.2016.09.006

Hosseini, S., & Barker, K. (2016). A Bayesian network model for resilience-based supplier selection. *International Journal of Production Economics*, 180, 68–87. https://doi.org/10.1016/j.ijpe.2016.07.007

Hosseini, S., & Ivanov, D. (2019). A new resilience measure for supply networks with the ripple effect considerations*:* a Bayesian network approach. *Annals of Operations Research*. https://doi.org/10.1007/s10479-019-03350-8

Hosseini, S., & Ivanov, D. (2020). Bayesian networks for supply chain risk, resilience and ripple effect analysis: A literature review. *Expert Systems with Applications*, 161, 113649. https://doi.org/10.1016/j.eswa.2020.113649

Hosseini, S., & Ivanov, D. (2021). A multi-layer Bayesian network method for supply chain disruption modelling in the wake of the COVID-19 pandemic. *International Journal of Production Research*, 1–19. https://doi.org/10.1080/00207543.2021.1953180

Hosseini, S., Ivanov, D., & Blackhurst, J. (2020). Conceptualization and measurement of supply chain resilience in an open-system context. *IEEE Transactions on Engineering Management*, 1–16. https://doi.org/10.1109/tem.2020.3026465

Hosseini, S., Ivanov, D., & Dolgui, A. (2019). Review of quantitative methods for supply chain resilience analysis. *Transportation Research Part E: Logistics and Transportation Review*, 125, 285–307. https://doi.org/10.1016/j.tre.2019.03.001

Jacobs, F. and Chase, R. (2000). *Administración de operaciones, producción y cadena de suministros* (13a. ed.). McGraw-Hill Interamericana.

Jacobs, F. R., & Chase, R. B. (2022). *Administración de operaciones: Producción y cadena de suministros* (Revisión técnica, 16.a ed.). McGraw-Hill Interamericana. Recuperado de https://www-ebooks7-24-com.bibliotecavirtual.uis.edu.co/stage.aspx?il=&pg=&ed=256

Jüttner, U., Peck, H., & Christopher, M. (2003). Supply chain risk management: outlining an agenda for future research. *International Journal of Logistics Research and Applications*, 6(4), 197–210. https://doi.org/10.1080/1367556031000162701 6

Koller, D., & Friedman, N. (2009). *Probabilistic graphical models: Principles and techniques.* (1st ed.). MIT Press.

Librantz, A. F. H., Costa, I., Spinola, M. de M., de Oliveira Neto, G. C., & Zerbinatti, L. (2021). Risk assessment in software supply chains using the Bayesian method. *International Journal of Production Research*, 59(22), 6758–6775. https://doi.org/10.1080/00207543.2020.1825860

Liang, X., Fan, S., Lucy, J., & Yang, Z. (2022). Risk analysis of cargo theft from freight supply chains using a data-driven Bayesian network. *Reliability Engineering & System Safety*, 226, 108702. https://doi.org/10.1016/j.ress.2022.108702

Liu, M., Liu, Z., Chu, F., Zheng, F., & Chu, C. (2020). A new robust dynamic Bayesian network approach for disruption risk assessment under the supply chain ripple effect. *International Journal of Production Research*, 59(1), 265–285. https://doi.org/10.1080/00207543.2020.1841318

Liu, N., Bouzembrak, Y., van den Bulk, L. M., Gavai, A., van den Heuvel, L. J., & Marvin, H. J. P. (2022a). Automated food safety early warning system in the dairy supply chain using machine learning. *Food Control*, 136, 108872. https://doi.org/10.1016/j.foodcont.2022.108872

Liu, M., Liu, Z., Chu, F., Dolgui, A., Chu, C., & Zheng, F. (2022b). An optimization approach for multi-echelon supply chain viability with disruption risk minimization. *Omega*, 112, 102683. https://doi.org/10.1016/j.omega.2022.102683

Lockamy, A., & McCormack, K. (2012). Modeling supplier risks using Bayesian networks. *Industrial Management & Data Systems*, 112(2), 313–333. https://doi.org/10.1108/02635571211204317

Losada, C., Scaparra, M. P., & O'Hanley, J. R. (2012). Optimizing system resilience: A facility protection model with recovery time. *European Journal of Operational Research, 217*(3), 519–530. https://doi.org/10.1016/j.ejor.2011.09.044

Manuj, I., & Mentzer, J. T. (2008). Global supply chain risk management strategies. *International Journal of Physical Distribution & Logistics Management*, 38(3), 192-223. https://doi.org/10.1108/09600030810866986

March, J. G., & Shapira, Z. (1987). Managerial perspectives on risk and risk taking. *Management Science*, 33(11), 1404–1418. https://doi.org/10.1287/mnsc.33.11.1404

Montgomery, D. C., & Runger, G. C. (1996). *Probabilidad y estadística aplicadas a la ingeniería*. McGraw-Hill.

Nan, C., & Sansavini, G. (2017). A quantitative method for assessing resilience of interdependent infrastructures. *Reliability Engineering and System Safety*, *157*, 35–53. http://dx.doi.org/10.1016/j.ress.2016.08.013

Ojha, R., Ghadge, A., Tiwari, M. K., & Bititci, U. S. (2018). Bayesian network modelling for supply chain risk propagation. *International Journal of Production Research, 56*(17), 5795-5819. https://doi.org/10.1080/00207543.2018.1467059

Pai, R., Kallepalli, V., Caudill, R. and Zhou, M. (2003). Methods toward supply chain risk analysis, *IEEE International Conference on Systems, Man and Cybernetics*, Vol. 5 No. 1, pp. 4560-5.

Pereda, M., & Zamarreno, J. M. (2015). Modelado basado en agentes: un enfoque desde la ingeniería de sistemas. Revista Iberoamericana de Automática e Informática Industrial, *12*(4), 304–312. http://dx.doi.org/10.1016/j.riai.2015.02.007

Ribeiro, J. P., & Barbosa-Povoa, A. (2018). Supply chain resilience: Definitions and quantitative modelling approaches – A literature review. *Computers & Industrial Engineering*, 115, 109–122. https://doi.org/10.1016/j.cie.2017.11.006

Rodgers, M., & Singham, D. (2019). A framework for assessing disruptions in a clinical supply chain using Bayesian belief networks. *Journal of Pharmaceutical Innovation*. https://doi.org/10.1007/s12247-019-09396-2

Ruskey, B., & Rosenberg, E. (2022). Minimizing risk in Bayesian supply chain networks. *Computers & Industrial Engineering*, 169, 108134. https://doi.org/10.1016/j.cie.2022.108134

Russell, S. and Norvig, P., (2003). *Inteligencia artificial. Un enfoque moderno*. Madrid: Pearson Educación.

Sakib, N., Ibne Hossain, N. U., Nur, F., Talluri, S., Jaradat, R., & Lawrence, J. M. (2021). An assessment of probabilistic disaster in the oil and gas supply chain leveraging Bayesian belief network. *International Journal of Production Economics*, 235, 108107. https://doi.org/10.1016/j.ijpe.2021.108107

Schildt, H. (2011). *Java: The complete reference* (8th ed.). Oracle Press.

Sichani, M. E., & Padgett, J. E. (2021). Performance assessment of oil supply chain infrastructure subjected to hurricanes. *Journal of Infrastructure Systems*, 27(4), 04021033. https://doi.org/10.1061/(ASCE)IS.1943-555X.0000637

Sharma, S. K., Routroy, S., & Chanda, U. (2022). Supply-side risk modelling using Bayesian network approach. *Supply Chain Forum: An International Journal*, 1–23. https://doi.org/10.1080/16258312.2021.1988697

Tang, C., 2006. Perspectives in supply chain risk management. *International Journal of Production Economics*, 103(2), pp.451-488. https://doi.org/10.1016/j.ijpe.2005.12.006

Wagner, S. M., & Bode, C. (2008). An empirical examination of supply chain performance along several dimensions of risk. *Journal of Business Logistics*, 29(1), 307-325. https://doi.org/10.1002/j.2158-1592.2008.tb00081.x

Wang, L., Ding, Y., & Wang, Y. (2022). A Bayesian Network Method for Humanitarian Supply Chain Performance Evaluation. *International Federation of Automatic Control*. https://doi.org/10.1016/j.ifacol.2022.10.203

Zhou, J., Shu-Ling (Peggy) Chen, Shi, W., Nguyen, S., Maneerat Kanrak, & Ge, J. (2023). A belief rule-based bayesian network approach for assessing risks in the cruise supply chain: An empirical study in Shanghai, China. 232, 106443–106443. https://doi.org/10.1016/j.ocecoaman.2022.106443

Apéndice A. Análisis Bibliométrico

Con el propósito de recopilar información científica relevante para la investigación, seleccionar documentos relacionados con la temática en estudio (redes bayesianas, análisis de riesgos y resiliencia en la cadena de suministro), y disponer de un insumo para la revisión de literatura que proporcione un contexto actualizado sobre la producción científica en el campo, se construyó una ecuación de búsqueda implementada en Scopus y Web of Science. Los documentos se identificaron de acuerdo con las siguientes palabras clave.

- Supply chain management OR supply chain
- Resilience OR risk
- Bayes* OR bayes model OR bayes* network*

Estas palabras clave fueron seleccionadas por su pertinencia al tema de investigación. Se combinaron de manera lógica para construir la ecuación de búsqueda, empleando operadores booleanos como "OR" y "AND", truncadores como el asterisco (*) en "Bayes" para capturar todas las variaciones posibles del término, así como comillas y paréntesis para estructurar la búsqueda. La ecuación se restringió a los campos de título, resumen y palabras clave, asegurando que los resultados obtenidos fueran relevantes para la investigación, evitando aquellos no relacionados con el tema central. Adicionalmente, la ventana de búsqueda se limitó a publicaciones comprendidas entre el año 2004 y diciembre de 2023.

La estructura de la ecuación implementada en cada herramienta es la siguiente:

Scopus: TITLE-ABS-KEY (("supply chain management" OR "supply chain") AND (resilience OR risk) AND (bayes* OR "bayes model" OR "bayes* network* ")) AND PUBYEAR > 2003 AND PUBYEAR < 2024 AND (LIMIT-TO (LANGUAGE, "English"))

Web of Science: TS=(("supply chain management" OR "supply chain") AND (resilience OR risk) AND (bayes* OR "bayes model" OR "bayes* network*")) AND LA= (English)

La búsqueda ***actualizada*** a diciembre de 2023 arrojó 266 registros en Scopus y 178 en Web of Science para un total de 444 (ver Tabla A1); sin embargo, algunos resultan repetidos si se comparan los dos resultados. Considerando lo anterior, se realiza el cruce de información, obteniendo 116 duplicados, y 328 documentos que permanecen

Tabla A1.

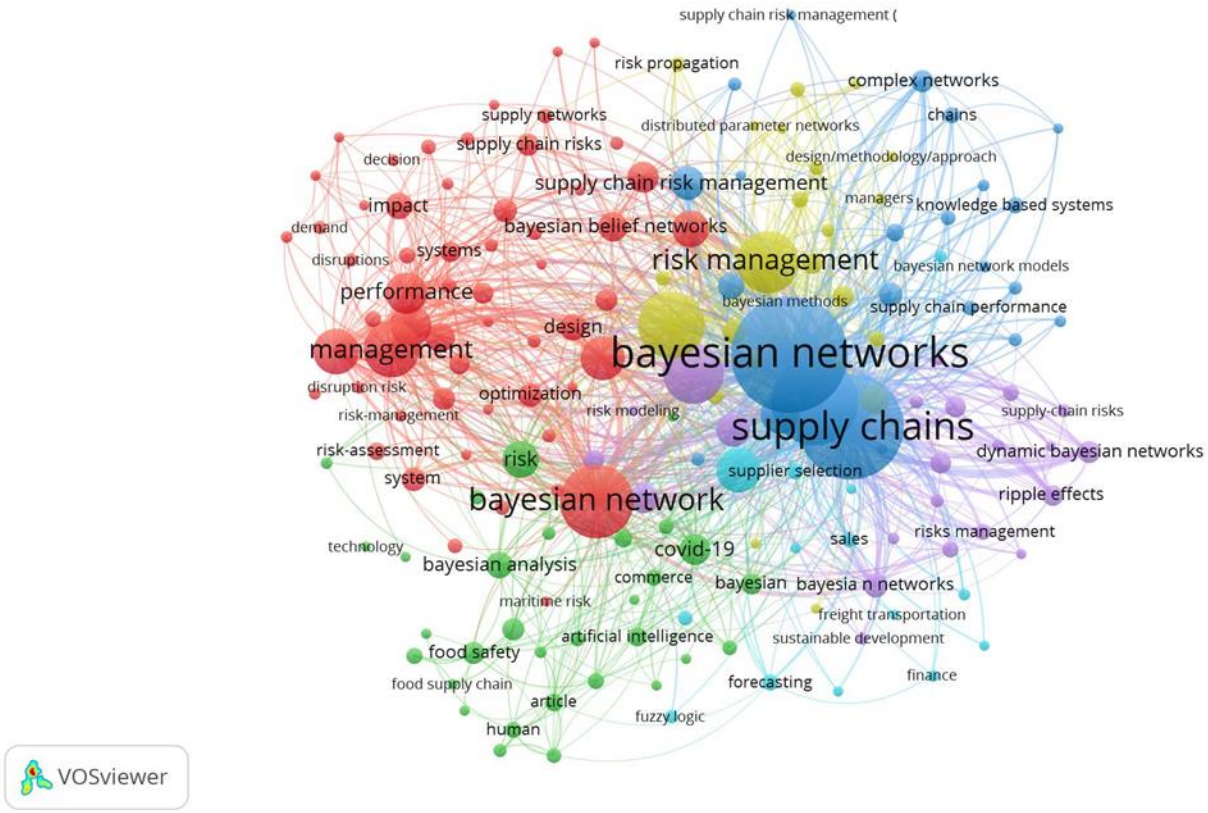
Resultados generales de información de Scopus y Web of Science

Base de datos	Article	Article; Early Access	Article; Proceedings Paper	Book	Book Chapter	Conference Paper	Conference Review	Review	Total
Scopus	153	0	0	1	7	79	20	6	**266**
Web Of Science	167	3	2	0	0	0	0	6	**178**
Total	**320**	**3**	**2**	**1**	**7**	**79**	**20**	**12**	**444**

A través del software VOSviewer, y tras depurar los duplicados, se llevó a cabo un análisis de coocurrencias de palabras clave (términos que los autores proporcionaron para describir el contenido principal de sus publicaciones o author keywords). Del conjunto de datos, con 2866 palabras clave en total (provenientes tanto de los registros de Scopus como Web Of Science) y considerando un mínimo de 5 coocurrencias, o que mínimo 5 veces la palabra clave debe aparecer en el conjunto de datos para ser incluida en el análisis, se encontró que 172 palabras cumplen con este umbral de frecuencia (número mínimo de coocurrencias). Estos términos principales se pueden visualizar en la Figura A1 que muestra la información en forma de red (solo se incluyen palabras clave que aparecen al menos 5 veces en el conjunto de documentos).

Figura A1.

Visualización de red de Keywords

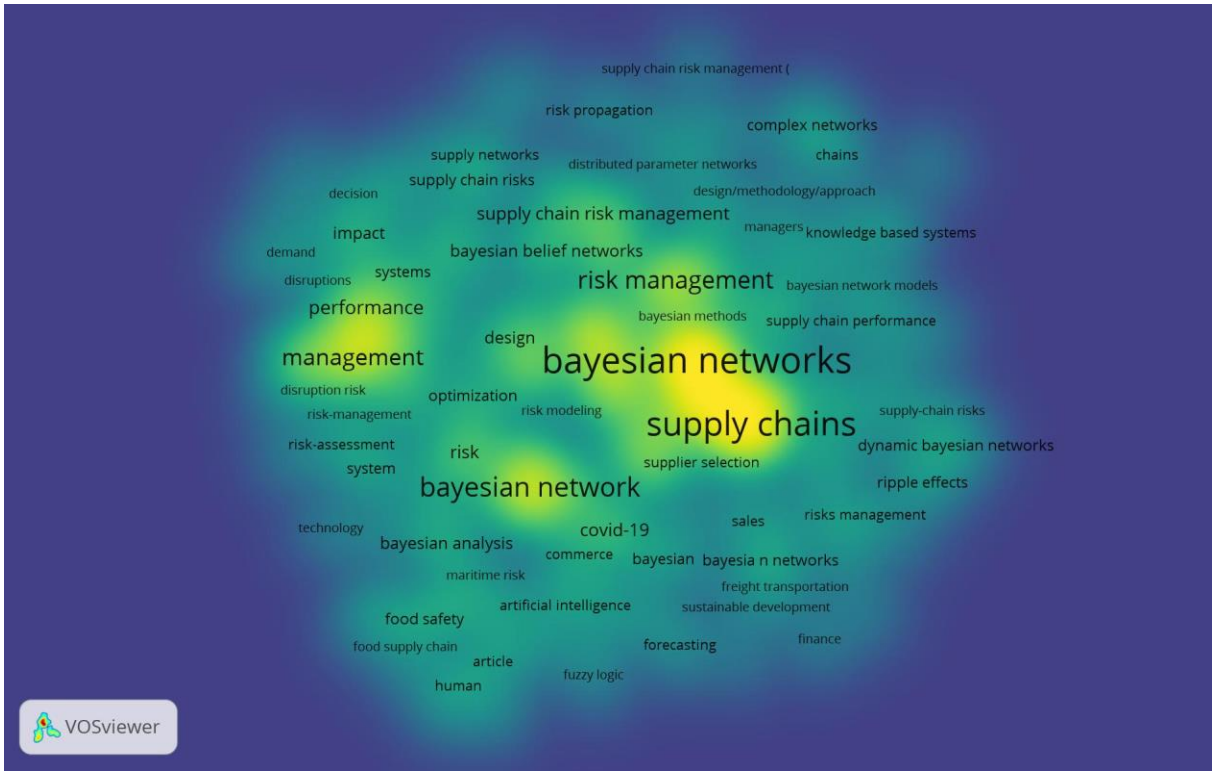


En la Figura A2 se muestra la información del análisis de coocurrencias, pero como un mapa de densidad; en él se pueden identificar las palabras clave más frecuentemente usadas o temas que tienen mayor protagonismo en la literatura o áreas más estudiadas. Los colores como el amarillo y el verde claro indican áreas de mayor densidad, lo que significa que esas palabras clave aparecen con mayor frecuencia en el conjunto de datos. Por ejemplo, términos como "bayesian networks" y "supply chains" están en el centro del mapa, resaltados en amarillo, lo que sugiere que son los temas más recurrentes. Por el contrario, los colores como el verde oscuro y el azul indican menor densidad, es decir, palabras clave que no aparecen tan frecuentemente, pero que aún son relevantes en el análisis.

Dado que la palabra resiliencia no aparece en el mapa de densidad, se puede interpretar que la resiliencia en la cadena de suministro, en el contexto de redes bayesianas, es un área aún en exploración o menos estudiada comparada con otros términos más recurrentes como "risk management" o "supply chain risks management".

Figura A2.

Mapa de densidad de Keywords



Por otro lado, una de las estadísticas relevantes en la producción científica sobre tema de investigación es la cantidad de publicaciones por año. En el caso de los 328 documentos que permanecen, esta información se presenta en la Tabla A2. Al graficar los datos en la Figura A3, se observa una tendencia creciente en la producción científica a lo largo del tiempo, siendo el 2022 el año con mayor número de publicaciones, con 58 documentos, lo que representa el 17,68% del total. En cuanto a 2023, también se registra una cifra elevada de publicaciones (14,02%).

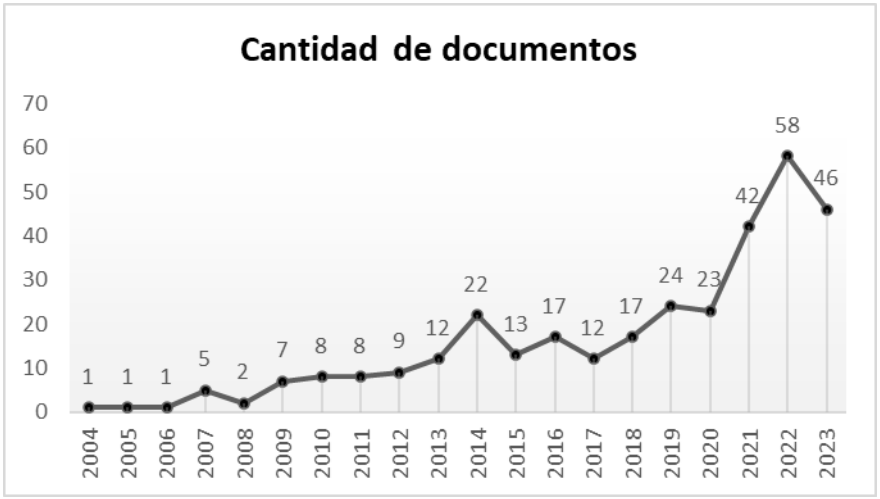
Tabla A2.

Año de publicación de los documentos

Año	Cantidad de documentos	Porcentaje
2004	1	0,30%
2005	1	0,30%
2006	1	0,30%
2007	5	1,52%
2008	2	0,61%
2009	7	2,13%
2010	8	2,44%
2011	8	2,44%
2012	9	2,74%
2013	12	3,66%
2014	22	6,71%
2015	13	3,96%
2016	17	5,18%
2017	12	3,66%
2018	17	5,18%
2019	24	7,32%
2020	23	7,01%
2021	42	12,80%
2022	58	17,68%
2023	46	14,02%
Total	**328**	**100%**

Figura A3.

Cantidad de documentos por año

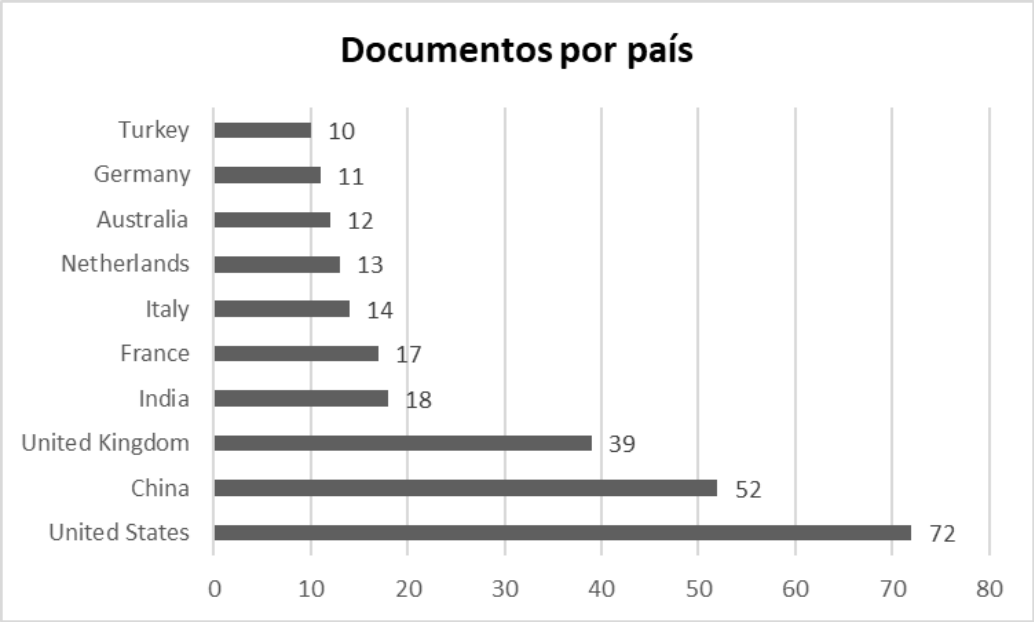


Con base en los resultados generados por Scopus, considerando sus registros, se realizan los siguientes análisis.

En la Figura A4 se muestran los 10 países con mayor cantidad de documentos en cuanto al tópico en estudio. Como se puede observar, Estados Unidos encabeza la lista con más de 70, seguido por China con 52 y Reino Unido con 39. Los siguientes países en la lista son India, Francia, Italia, Países Bajos, Australia, Alemania y Turquía.

Figura A4.

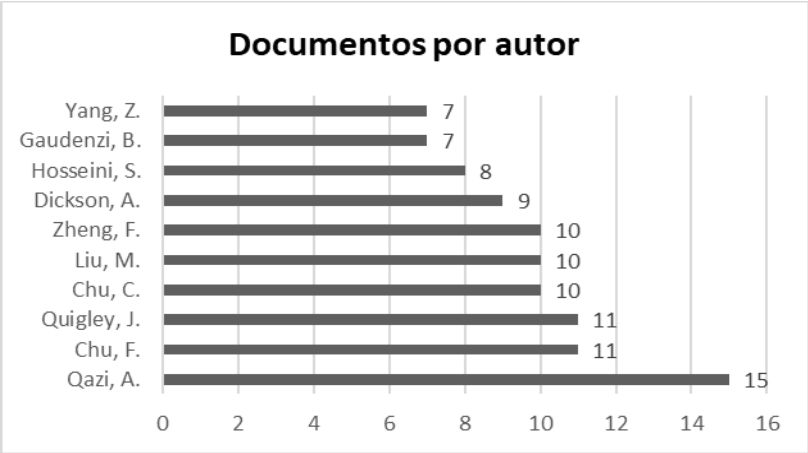
Cantidad de documentos por país



En la Figura A5 se presentan los principales autores, destacándose Qazi, A., con el mayor número de publicaciones, seguido por Chu, F., y Quigley, J. También se destacan Chu, C., Liu, M., Zheng, F., Dickson, A., y Hosseini, S., entre otros.

Figura A5.

Documentos por autor



Finalmente, la Tabla A3 muestra los 10 investigadores con mayor número de documentos publicados y sus correspondientes citaciones acumuladas. Como se observa en la Tabla 9, **Qazi, A.** lidera en cuanto a número de documentos, con un total de 15 publicaciones. Sin embargo, en términos de citaciones, ocupa una posición intermedia con 300. Esto indica que, aunque su productividad es alta, su impacto, medido a través de citaciones, es relativamente menor en comparación con **Hosseini, S.,** quien se destaca significativamente con un total de 936 citaciones. Por otro lado, **Quigley, J.** también presenta una cantidad intermedio de citaciones, alcanzando las 313.

Tabla A3.

Documentos y citaciones por autor

Autor	Documentos por autor	Citaciones
Qazi, A.	15	300
Chu, F.	11	95
Quigley, J.	11	313
Chu, C.	10	94
Liu, M.	10	94
Zheng, F.	10	95
Dickson, A.	9	267
Hosseini, S.	8	936
Gaudenzi, B.	7	188
Yang, Z.	7	433

Apéndice B. Gestión de Inventarios

Un sistema de control de inventarios establece las políticas operativas para mantener y controlar los bienes que se almacenarán, siendo responsable de su pedido y recepción (Jacobs & Chase, 2022). El objetivo de un sistema de control de inventarios es determinar cuándo y cuánto pedir, y la decisión de pedido no puede basarse solo en el stock disponible, sino considerar los pedidos pendientes que aún no han llegado (Axsäter, 2006).

B1. Modelos de Gestión de Inventario

Jacobs y Chase (2022) dividen los sistemas de control de inventarios en sistemas de periodo único y sistemas de periodos múltiples. En el sistema de periodo único, el artículo que se comprará es una única compra para cubrir un periodo fijo. En contraste, en los sistemas de periodos múltiples, se compran periódicamente los artículos para ser utilizados bajo demanda, y se establecen tanto la cantidad a ordenar como el momento en que se debe realizar el pedido.

En cuanto a los sistemas de inventario de periodos múltiples, existen dos modelos:

- Modelos de cantidad fija de pedido también denominado cantidad económica de pedido (EOQ, Economic Order Quantity) o modelo Q.
- Modelos de periodo fijo, también conocidos como sistema periódico, sistema de revisión periódica, sistema de intervalo de pedido fijo o modelo P (Jacobs & Chase, 2022).

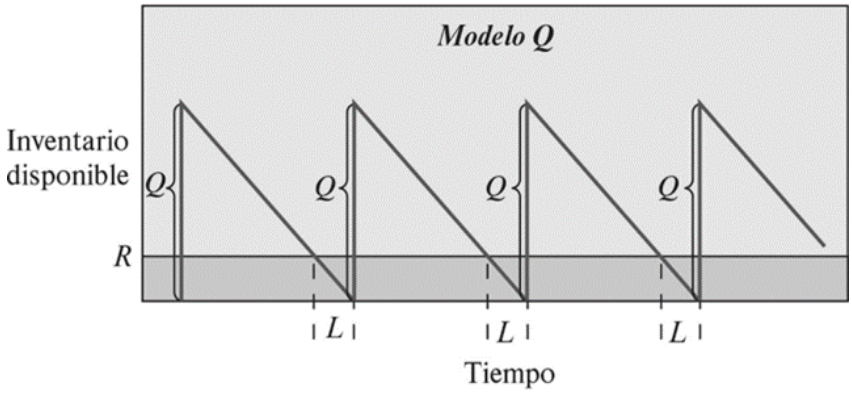
B1.1 Modelos de Cantidad Fija de Pedido

En este tipo de modelos, se define un punto específico, denominado R, en el que se debe realizar un pedido, así como el tamaño de dicho pedido (Q). La decisión de cuándo ordenar se toma en relación con este punto de pedido R, también conocido como ROP (Reorder Point o punto de reorden), que representa el nivel de inventario o la posición de inventario en la que debe

colocarse el pedido. La Figura B1 ilustra el funcionamiento del modelo, mostrando que, cuando la posición del inventario cae al punto R, se debe colocar una orden. Se coloca la orden de tamaño Q cuando la posición de inventario (que para este modelo básico solo incluye el inventario en existencia y el que está en tránsito) alcanza el punto R. Este pedido se recibe al final del periodo L, que en este caso es constante (Jacobs & Chase, 2022).

Figura B1.

Modelo básico de la cantidad fija de pedido



Adaptado de *Administración de operaciones: Producción y cadena de suministros*, por F. R. Jacobs y R. B. Chase, 2022, McGraw-Hill Interamericana.

Este modelo básico no considera inventario de seguridad, ya que asume:

- Una demanda constante y uniforme durante todo el periodo.
- Un tiempo de entrega constante.
- Que todas las demandas serán satisfechas (no se permiten pedidos pendientes) (Jacobs & Chase, 2022).

Luego el punto de reorden ROP es:

$$ROP = d \times L$$

Esta ecuación supone que la demanda durante el tiempo de entrega y el tiempo de entrega son constantes.

Donde:

$$d = Demanda\ diaria\ promedio\ constante$$

$$L = Tiempo\ de\ entrega\ en\ días\ constante$$

B1.2 Modelos de Periodo Fijo

En este tipo de modelos, el inventario se revisa únicamente en intervalos específicos, como semanal o mensualmente. A diferencia de los modelos de cantidad fija, aquí los pedidos se realizan de manera periódica, y las cantidades solicitadas pueden variar de un periodo a otro. Los modelos estándar de periodo fijo suponen que el inventario solo se cuenta en el momento especificado para su revisión (Jacobs & Chase, 2022).

B2. Clasificación de Políticas de Pedido

Por otra parte, Axsäter (2006) realiza una clasificación diferente en lo que respecta al control de inventarios, señalando que las dos políticas de pedido más comunes son las denominadas política (R, Q) y política (s, S) (Axsäter, 2006).

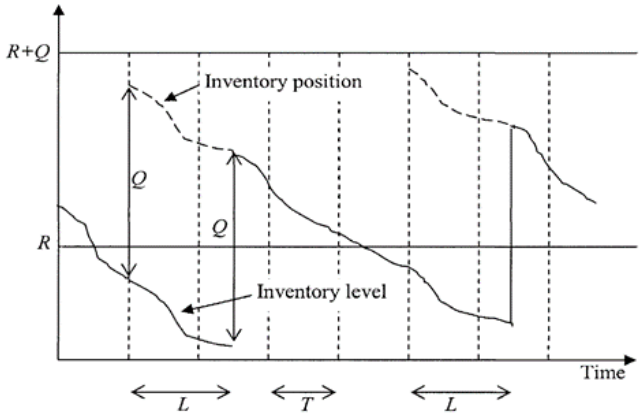
B2.1 Política $(\mathbf{R}, \mathbf{Q})$

En esta política, cuando la posición de inventario disminuye hasta o por debajo del punto de reorden R, se pide una cantidad fija Q. Si la demanda es continua, o de una unidad a la vez y, el inventario se revisa constantemente (revisión continua), se garantiza que siempre se realice un pedido justo al alcanzar el punto de reorden, evitando que el inventario caiga por debajo de ese nivel (Axsäter, 2006).

En el caso de revisión periódica, o cuando la demanda que activa el pedido es superior a una unidad, es probable que la posición de inventario se encuentre por debajo del punto de reorden en el momento de realizar el pedido, situación que se puede visualizar en la Figura B2.

Figura B2.

Política (R, Q) con revisión periódica. Demanda continua



Tomado de *Inventory Control*, por S. Axsäter, 2006, Springer Science+Business Media.

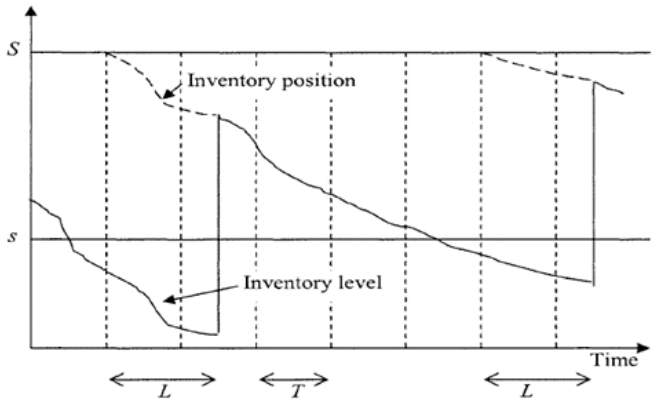
En resumen, la política (R, Q) establece que se realiza un pedido cuando el inventario baja a o se encuentra por debajo del punto de reorden R; el pedido consiste en una cantidad fija Q (Axsäter, 2006).

B2.2 Política (s, S)

En este tipo de política, el punto de reorden se denota por "s", y se realiza un pedido cuando la posición del inventario baja hasta este punto o menos. La cantidad a ordenar es la necesaria para reestablecer la posición de inventario hasta el nivel máximo S, (Axsäter, 2006).

Figura B3.

Política (s, S) con revisión periódica. Demanda continua



Tomado de *Inventory Control*, por S. Axsäter, 2006, Springer Science+Business Media

La Figura B3 ilustra cómo funciona una política (s, S) con revisión periódica y demanda continua. Dado que se emplea revisión periódica, es posible que se ordene cuando la posición de inventario esté por debajo del punto de reorden, y que al recibir el pedido no se alcance el nivel máximo S (Axsäter, 2006).

En este sentido, la política (R, Q) es equivalente a la (s, S) si se considera revisión continua y demanda continua; es decir, se ordena exactamente en el punto de reorden, y siempre que "$s = R$" y "$S = R + Q$". En caso contrario, la equivalencia no se mantiene, ya que en revisión periódica puede suceder que, con cualquiera de las dos políticas, no se alcance el punto de reorden en una determinada inspección y no se active el pedido (Axsäter, 2006).

Una variación de la política (s, S), es conocida como política S, o política de pedido hasta S (order up to-S), o política de stock base. Esta variación se caracteriza porque siempre se ordena, a menos que la demanda del período sea cero. Es decir, una política S simplemente significa que se ordena hasta S, independientemente de la posición del inventario. Cada vez que se revisa el inventario, se realiza un pedido para elevarlo hasta S. A diferencia de la política (s, S), en la cual solo se realiza un pedido cuando el nivel de inventario cae hasta o por debajo del punto de reorden, denotado por "s". Esta política es equivalente a una política (s, S) en la que $s = S - 1$; es decir, se reordena cuando el inventario está en $S - 1$ o menos, y se hace el pedido hasta alcanzar S (Axsäter, 2006).

B3. Modelos Probabilísticos e Inventario de Seguridad

Los modelos de inventario anteriores asumían que la demanda del producto es constante y cierta. Sin embargo, según Heizer y Render (2009), si se elimina este supuesto y la demanda del producto no se conoce, o si los tiempos de entrega tampoco se conocen, pero pueden especificarse

a través de una distribución de probabilidad, se está ante modelos denominados “modelos probabilísticos” (Heizer & Render, 2009).

La demanda incierta incrementa la posibilidad de faltantes. Un método adecuado para disminuir la cantidad de faltantes es mantener inventario adicional, comúnmente denominado “inventario de seguridad”, que implica sumar cierto número de unidades al punto de reorden, actuando como un amortiguador. (Heizer & Render, 2009).

La inclusión del inventario de seguridad (ss) cambia la expresión del punto de reorden a:

$$ROP = (d \times L) + ss$$

Se puede decidir establecer una política que mantenga un inventario de seguridad suficiente para satisfacer un determinado nivel de servicio al cliente. Por ejemplo, se podría definir el nivel de servicio en satisfacer el 95% de la demanda (o tener faltantes sólo un 5% del tiempo). Si se supone que durante el tiempo de entrega la demanda sigue una distribución normal, sólo se necesitan la media y la desviación estándar para definir los requerimientos de inventario en cualquier nivel de servicio. En general, los datos de ventas son adecuados para calcular la media y la desviación estándar (Heizer & Render, 2009).

Por ejemplo, retomando a Jacobs y Chase (2022), en un modelo de cantidad fija de pedido donde la demanda es incierta (ver Figura B4) y se distribuye normalmente, el punto de reorden se calcula agregando la cantidad $Z \times \sigma_{dLT}$ que se conoce como inventario o existencias de seguridad, de la siguiente manera:

$$ROP = (d \times L) + (Z \times \sigma_{dLT})$$

Demanda esperada durante el tiempo de entrega

Donde:

$d = Demanda\ media\ diaria$

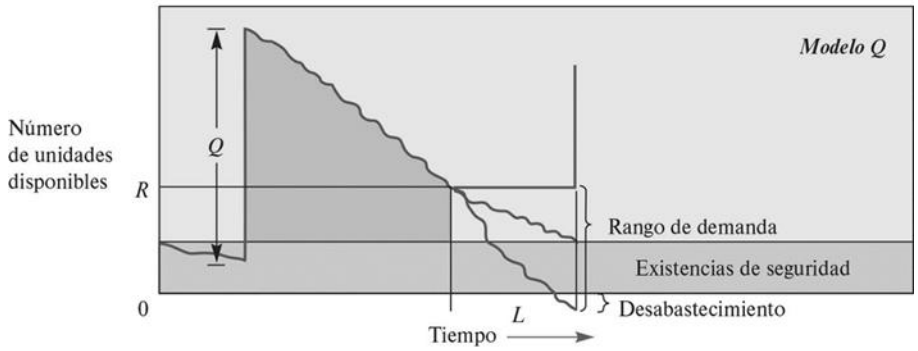
$L = Tiempo\ de\ entrega\ en\ días$

$Z = \#\ de\ desviaciones\ estandar\ para\ una\ probabilidad\ de\ servicio\ especificada$

$\sigma_{dLT} = Desviación\ estándar\ de\ la\ demanda\ durante\ el\ tiempo\ de\ entrega$

Figura B4.

Modelo de la cantidad fija de pedido con incertidumbre en la demanda

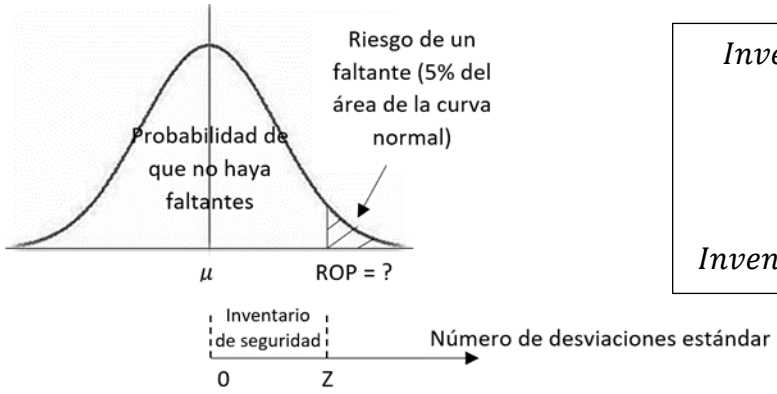


Adaptado de *Administración de operaciones: Producción y cadena de suministros*, por F. R. Jacobs y R. B. Chase, 2022, McGraw-Hill Interamericana.

El inventario de seguridad, es calculado como Z × σ_{dLT} porque, si la demanda es incierta, representada por una curva normal con media ($\boldsymbol{\mu}$) y desviación estándar ($\boldsymbol{\sigma}$) conocidas, la cantidad necesaria para alcanzar un determinado nivel de servicio es ilustrada en la Figura B5, donde se observa la curva normal de la demanda.

Figura B5.

Curva normal de demanda para cálculo de ss



$$Inventario\ de\ seguridad = x - \mu$$

Y: $Z = \frac{x - \mu}{\sigma_{dLT}}$ Entonces

$$Inventario\ de\ seguridad = Z \times \sigma_{dLT}$$

Adaptado de *Principios de administración de operaciones*, por J. Heizer y B. Render, 2009, Pearson Educación.

Sin embargo, cuando el tiempo de entrega también es incierto, la fórmula anterior no puede aplicarse, y es necesario determinar qué modelo utilizar. En estos casos, donde tanto el tiempo de entrega como la demanda son variables, la fórmula para el punto de reorden se vuelve más compleja (Heizer & Render, 2009).

$$ROP = (\bar{d} \times Tprom) + Z\sigma_{total}$$

Donde:

$\bar{d} = Demanda\ diaria\ promedio$

$Tprom = Tiempo\ de\ entrega\ promedio$

$Z = Nivel\ de\ confianza$

$\sigma_{total} = Desviación\ estándar\ total.$ Reflejando la variabilidad combinada de la demanda y del tiempo de entrega.

Cuando se trata de combinar incertidumbres de dos fuentes (demanda y tiempo de entrega), se suman las varianzas para obtener la varianza total. Luego, para obtener la desviación estándar total, se toma la raíz cuadrada de esa suma. La fórmula resultante refleja el efecto combinado de la variabilidad tanto en la demanda diaria como en el tiempo de entrega (Heizer & Render, 2009):

$$\sigma_d = Desviación\ estándar\ de\ la\ demanda\ diaria$$

$$\sigma_{LT} = Desviación\ estándar\ del\ tiempo\ de\ entrega\ en\ días$$

$$\sigma_{total} = \sqrt{(Tprom \times \sigma_d^2) + \bar{d}^2\sigma_{LT}^2}$$

Luego:

$$ROP = (\bar{d} \times Tprom) + Z\sqrt{(Tprom \times \sigma_d^2) + \bar{d}^2\sigma_{LT}^2}$$

Apéndice C. Explicación de métodos de cada una de las clases del modelo en Python

C1. Métodos de la clase minorista. Esta clase incluye varios métodos que permiten gestionar la operación de un objeto minorista en la cadena de suministro. El primero es el **constructor** (__init__) para inicializar los atributos del objeto cuando se crea una instancia de la clase. Configura parámetros como el nombre del minorista, los datos sobre la media y desviación estándar de la demanda diaria (media_demanda, desv_demanda), el tiempo mínimo y máximo de entrega que es el número de días mínimo y máximo que el cliente tarda en recibir los productos después de despachados (min_tiempo_entrega y max_tiempo_entrega), el tiempo mínimo y máximo de producción, establecidos en 0 porque los minoristas no tienen proceso de manufactura (min_tiempo_pcc y min_tiempo_pcc), el nivel de inventario objetivo de productos terminados que el minorista intenta mantener (nivel_objetivo), el inventario de productos en proceso, también en 0 porque no tiene proceso de manufactura (stock_producto_en_proceso_MINO), el inventario de producto terminado listo para la venta (stock_producto_TERMINADO_minorista), el límite máximo de producción diario, en 0 por ausencia de proceso de manufactura (max_limite_pcc).

Adicionalmente, en el constructor se definen otros atributos y listas como:

Una lista (lista_venta_pendiente) para almacenar un diccionario con la cantidad de unidades que no se pudo satisfacer en el día debido a falta de stock “unidades_de_venta_pendiente”. Estas unidades se almacenan junto con un contador de días de espera “dias_espera”, inicialmente establecido en 0.

Una variable booleana o bandera (dia_sin_stock_mino) que se usa para rastrear si el minorista tiene suficiente stock para procesar todas las órdenes del día. Inicializada como “False al comienzo de iniciar la revisión diaria de pedidos pendientes, lo que significa que no se tiene

falta de stock. Esta variable cambia a “True” si al barrer los pedidos pendientes no hubo suficiente stock para cubrir la demanda.

Una lista (historial_dias_sin_stock_mino) que almacena la variable dia_sin_stock_mino de para cada uno de los días de la réplica, a fin de disponer del historial.

Una variable (contador_dias_sin_stock) que cuenta los días en los que hubo falta de stock. Inicializada en 0.

El punto_de_reorden (punto_de_reorden), que indica el nivel de inventario en el que el minorista debe reordenar productos (inicialmente establecido en 0)

La cantidad de productos que el minorista espera recibir del mayorista (cantidad_PENDIENTE_a_recibir_del_mayorista), inicialmente establecida en 0

Una variable (dias_para_recibir) relacionada con el tiempo de entrega que tarda en llegar un pedido del mayorista, luego de despachado. Inicialmente establecida en 0. Se calcula usando el método calculo_tiempo_entrega() de la instancia mayorista.

Una variable (total_ventas_pendientes_CLIENTE) que se utiliza para almacenar el total de unidades de venta pendientes que aún no han sido cumplidas debido a la falta de stock en el minorista. Esta variable se actualiza en el método procesar_venta_pendiente, donde se suman todas las unidades que aún están pendientes de ser vendidas (es decir, que los clientes solicitaron pero no pudieron recibir inmediatamente por falta de producto). Cada vez que se ejecuta procesar_venta_pendiente, este total se recalcula sumando las ventas pendientes actuales. El objetivo de este atributo es llevar un registro del número total de productos que los clientes aún están esperando recibir.

Los demás métodos del minorista son:

Un método (**calculo_tiempo_entrega**) para simular la variabilidad en los tiempos de entrega de un pedido del minorista, usando una distribución uniforme entre el tiempo mínimo y máximo de entrega. El cliente recibe los productos inmediatamente son despachados por el minorista.

Un método (**demanda_venta**) para simular la demanda de los clientes. Utiliza una distribución normal basada en la media y desviación estándar de la demanda para generar un número de unidades demandadas (demandaMino). La venta efectiva está limitada según el stock de producto terminado en el almacén (stock_producto_TERMINADO_minorista), es decir, si la demanda supera el inventario de producto terminado, se venden únicamente las unidades que están en el inventario de producto terminado. Se actualiza el inventario (stock_producto_TERMINADO_minorista) restando las ventas realizadas (venta_efectiva). Si hay más demanda que producto terminado es decir demanda insatisfecha (venta_perdida = max(0, demandaMino - venta), se registra la cantidad de venta pendiente en la lista (lista_venta_pendiente), junto con un contador de días en espera (dias_espera). El cliente sólo está dispuesto a esperar 1 día.

Un método (**procesar_venta_pendiente**) que se encarga de procesar la cantidad de venta pendiente del día anterior. Este método comienza inicializando una lista vacía denominada (nueva_lista_venta_pendiente), y el atributo (dia_sin_stock_mino) en “False”, luego para la venta en la lista_venta_pendiente, verifica si se tiene suficiente stock para cubrirla. Si lo hay, la venta se satisface y se reduce el stock_producto_TERMINADO_minorista en la cantidad vendida. Si no es posible cumplir la venta (no se tiene suficiente stock), se incrementa el contador de (dias_espera) en 1 y se activa la bandera o variable booleana (dia_sin_stock_mino) cambiando su valor a “True” que indica que el minorista se quedó sin stock ese día para satisfacer la venta. Si un cliente ya ha

esperado un día completo (venta["dias_espera"] == 1), se considera que no espera más y se pierde definitivamente la venta. Después, se cambia la lista_venta_pendiente por la nueva_lista_venta_pendiente, y se calcula el total de todas las unidades de venta que aún están pendientes, almacenando ese valor en la variable self.total_ventas_pendientes_CLIENTE. Posteriormente se actualiza la lista que almacena el historial de días sin stock (historial_dias_sin_stock_mino), así como el contador que cuenta los días en los que hubo falta de stock (contador_dias_sin_stock).

Un método (**calculo_cantidad_a_ordenar_al_mayorista**) que se encarga de calcular la cantidad que el minorista debe ordenar al mayorista. Para ello, se determina el promedio del tiempo de entrega del mayorista (T_promedio), tomando en cuenta que los tiempos de entrega siguen una distribución uniforme, donde cualquier valor entre los límites mínimo (min_tiempo_entrega) y máximo (max_tiempo_entrega) es igualmente probable. Además, se calcula la desviación estándar del tiempo de entrega (sigma_LT), que, para una variable aleatoria uniforme, se obtiene dividiendo la diferencia entre el máximo y el mínimo por la raíz cuadrada de 12. Así mismo, se calcula la variabilidad total (sigma_total) del tiempo de entrega y la demanda, y se establece un nivel de confianza del 95% para un valor de Z de 1.96. Esto con el fin de calcular el punto de reorden que incluye tanto la incertidumbre de la demanda como la del tiempo de entrega. Una vez establecido el punto de reorden. Si la posición de inventario (self.posicion_inventario) es menor o igual al punto de reorden, se calcula la cantidad que debe ordenarse (cantidad_ordenada_mino) restando la posición de inventario del nivel objetivo (self.nivel_objetivo - self.posicion_inventario). Si la posición de inventario es mayor que el punto de reorden, no se necesita realizar pedido, y se establece cantidad_ordenada_mino en 0. Finalmente, el método devuelve la cantidad a ordenar.

Un método (**generar_orden_al_mayorista**) que genera la orden de compra al mayorista, según la cantidad a ordenar del método calculo_cantidad_a_ordenar_al_mayorista. Para ello, se define un diccionario vacío llamado orden_minorista el cual almacena los detalles de la orden. Se verifica si la cantidad a ordenar (cantidad_ordenada_mino) es mayor que cero. Si es así, se calcula el tiempo de entrega (dias_para_recibir) usando el método calculo_tiempo_entrega() del objeto mayoristaR que calcula el tiempo estimado para que la orden llegue desde el mayorista al minorista luego de ser despachada. Se crea el diccionario (orden_minorista) con detalles de la información del pedido, asignando: A la clave 'minorista' el nombre del minorista, valor de (self.nombre). A la clave 'cantidad' el valor de (cantidad_ordenada_mino). A la clave 'dias_tiempo_entrega' el valor de (dias_para_recibir). A la clave 'mayorista' el valor de (nombre_mayorista). Luego, se actualiza la cantidad pendiente que el minorista espera recibir del mayorista (self.cantidad_PENDIENTE_a_recibir_del_mayorista), sumando la cantidad_ordenada_mino. Finalmente, la orden generada se envía al mayorista (mayoristaR.recibe_orden_del_minorista(orden_minorista)). Llamando al método recibe_orden_del_minorista() del objeto mayoristaR, pasándole la orden generada (orden_minorista).

Por último, la clase minorista tiene un método (**recibe_producto_del_mayorista**) que define el comportamiento del minorista al recibir productos del mayorista. Actualizando el atributo (self.stock_producto_TERMINADO_minorista), que es el inventario de producto terminado del minorista, mediante la suma de la cantidad de productos que ha enviado el mayorista (cantidad_producto_enviadoMinoF). Al finalizar, se resta la cantidad de producto recibida (cantidad_producto_enviadoMinoF) de la cantidad total que el minorista tiene pendiente recibir

del mayorista (self.cantidad_PENDIENTE_a_recibir_del_mayorista). Esto significa que ahora el minorista tiene menos productos por recibir del mayorista.

C2. Métodos de la clase mayorista. Los métodos de esta clase permiten gestionar la operación de un objeto mayorista en la cadena de suministro. El primero es el **constructor** (__init__) para inicializar los atributos del objeto cuando se crea una instancia de la clase. Configura parámetros como el nombre del mayorista, los datos sobre la media y desviación estándar de la demanda diaria (media_demanda, desv_demanda), el tiempo mínimo y máximo de entrega al minorista (min_tiempo_entrega y max_tiempo_entrega), el tiempo mínimo y máximo de producción, establecidos en 0 porque los mayoristas no tienen proceso de manufactura (min_tiempo_pcc y min_tiempo_pcc), el stock objetivo de productos terminados que el mayorista intenta mantener (nivel_objetivo), el inventario de productos en proceso (stock_producto_en_proceso_MAYO), también en 0 porque no tiene proceso de manufactura, el inventario de productos terminados listos para ser despachados (stock_producto_TERMINADO_mayorista), el límite máximo de producción diario, en 0 por ausencia de proceso de manufactura (max_limite_pcc).

Adicionalmente, se definen otros atributos y listas como:

Una lista (ordenes_pendientes_MINO) para almacenar todas las órdenes que el mayorista ha recibido del minorista pero que aún no han sido completamente procesadas o entregadas. Una orden se considera procesada cuando el mayorista ha disminuido el inventario en la cantidad de la orden, pero los productos aún están en tránsito esperando a ser entregados, y una orden se considera entregada cuando ha llegado a su destino; es decir, el minorista finalmente recibe los productos. Esto ocurre después de un período de tiempo, representado por dias_tiempo_entrega. El código controla este período, y cuando dias_tiempo_entrega llega a 0, la orden se entrega al minorista.

Una lista (nuevas_ordenes_pendientes_MINO) que se utiliza como lista temporal donde se colocan las órdenes que, después del ciclo de procesamiento todavía deben estar pendientes (ya sea porque no se han procesado o porque aún les quedan días de entrega. Esta lista reemplaza a la lista ordenes_pendientes_MINO al finalizar el ciclo de procesamiento, para que en el próximo día se siga trabajando con las órdenes que aún están pendientes. Su función es realizar un seguimiento de las órdenes que deben mantenerse en la lista de ordenes_pendientes_MINO para el siguiente ciclo de procesamiento.

La variable booleana (dia_sin_stock_mayo) para rastrear si el mayorista tiene suficiente stock para procesar todas las órdenes pendientes del día. Inicializada como “False” al comienzo de iniciar la revisión diaria de pedidos pendientes, lo que significa que no se tiene falta de stock. Esta variable cambia a “True” si al barrer los pedidos pendientes no hubo suficiente stock para cubrir la demanda.

La lista (historial_dias_sin_stock_mayo) que almacena la variable dia_sin_stock_mayo de cada uno de los días de la réplica, a fin de disponer del historial.

El contador de días sin stock (contador_dias_sin_stock) que cuenta los días en los que hubo falta de stock. Inicializado en 0.

El punto_de_reorden (punto_de_reorden), que indica el nivel de inventario en el que el mayorista debe reabastecerse (inicialmente establecido en 0)

La cantidad de productos que el mayorista espera recibir de los productores (cantidad_PENDIENTE_a_recibir_de_los_productores)

Una variable (dias_para_recibir) relacionada con el tiempo de entrega que tarda en llegar un pedido del productor, luego de despachado. Inicialmente establecida en 0. Se calcula usando el método calculo_tiempo_entrega() de la instancia productor.

La variable self.total_pedidos_pendientes_MINO que almacena el total de productos que están pendientes de ser procesados por el mayorista, es decir, aquellos pedidos que están es estado NO_PROCESADA (aún no han sido despachados a los minoristas).

Los demás métodos incluyen:

Un método (**calculo_tiempo_entrega)** para calcular el tiempo de entrega de un pedido, que genera un número aleatorio dentro de los límites definidos, usando los valores mínimo y máximo.

Un método (**recibe_orden_del_minorista**) que se encarga de recibir la orden de pedido del minorista y almacenarla en una lista de órdenes pendientes (self.ordenes_pendientes_MINO) sin procesarla inmediatamente. El método recibe como parámetro un diccionario llamado orden_minorista, que contiene claves que le fueron asignadas como el nombre del minorista que envía la orden 'minorista', la cantidad de productos que el minorista solicita 'cantidad', los 'dias_tiempo_entrega', el tiempo estimado de entrega para cumplir la orden después de ser PROCESADA y el nombre del mayorista a quien va dirigida 'mayorista'. Este método recibe_orden_del_minorista inicialmente extrae la información del diccionario, y la asigna a las variables locales.

- nombre_minorista_recibido_por_mayo: Toma el valor asociado a la clave 'minorista', que indica el nombre del minorista que ha enviado la orden.
- cantidad_ordenada_mino_recibido_por_mayo: Extrae el valor de la clave 'cantidad', que especifica cuántos productos ha solicitado el minorista.
- dias_tiempo_entrega_mayo: Obtiene el valor de la clave 'dias_tiempo_entrega', que representa los días estimados de tiempo de entrega.
- nombre_mayorista_recibido_por_mayo: Extrae el valor de la clave 'mayorista', es decir, el nombre del mayorista a quien va dirigida la orden

Posteriormente, el método almacena la orden en la lista ordenes_pendientes_MINO en 'estado' de 'NO_PROCESADA'. Para ello crea un nuevo diccionario con las claves 'minorista': que guarda el nombre del minorista que hizo la solicitud; 'cantidad': que guarda la cantidad de productos solicitados; 'dias_tiempo_entrega': que almacena el tiempo estimado en días para que el mayorista después de procesada la orden la entregue; 'mayorista': que guarda el nombre del mayorista que recibe la orden de pedido; el 'estado': marcado como 'NO_PROCESADA', que indica que aún no ha sido atendida; y finalmente, los 'dias_espera': un contador inicializado en 0 que sirve para registrar cuántos días han pasado desde que la orden de compra fue recibida por el mayorista.

Un método (**mayorista_procesa_ordenes_pendientes_del_minorista**) cuyo propósito es tratar de procesar todas las órdenes pendientes del minorista, de acuerdo al inventario de productos terminados. Para comenzar, se inicializa la bandera (self.dia_sin_stock_mayo) en “False”, utilizada para verificar si durante el día, el mayorista se quedó sin stock suficiente para procesar las órdenes. Luego, para cada orden de la lista (ordenes_pendientes_MINO) aumenta en uno el número de días que ha estado esperando cada orden (orden['dias_espera'] += 1). Verifica si la orden está en estado "NO_PROCESADA", y si el stock del mayorista (stock_producto_TERMINADO_mayorista) es suficiente para cubrir la cantidad completa de la orden, si sí, se reduce el inventario en la cantidad respectiva, y la orden se marca como "PROCESADA", pero si el stock del mayorista no es suficiente para cumplir con toda la cantidad solicitada y aún queda algo de stock, se procesa parcialmente la orden. En este caso: La cantidad que se puede despachar (cantidad_despachada = self.stock_producto_TERMINADO_mayorista) se resta del stock disponible, y se genera una nueva orden con la cantidad restante no procesada. Esta nueva orden generada se agrega a la lista de ordenes_pendientes_MINO justo después de la orden actual. Se marca la orden actual como "PROCESADA" y se establece la bandera

dia_sin_stock_mayo en True para indicar que no se pudo satisfacer completamente. Pero, si el mayorista no tiene suficiente stock para despachar ninguna cantidad de la orden, se marca la bandera dia_sin_stock_mayo como True y se continúa con la siguiente orden. Al finalizar la iteración, se guarda en el historial si el mayorista tuvo o no problemas de stock ese día, añadiendo el valor de dia_sin_stock_mayo al historial de días sin stock (historial_dias_sin_stock_mayo). El contador de días sin stock (contador_dias_sin_stock) también se incrementa en 1 si el mayorista experimentó al menos una orden no procesada por falta de stock durante el día (dia_sin_stock_mayo es True).

Después, se recorre cada orden de la lista self.ordenes_pendientes_MINO, verificando los días restantes para su entrega. Si una orden tiene dias_tiempo_entrega igual a 0 y está marcado como 'PROCESADA', se extrae la cantidad de producto a enviar y el nombre del minorista asociado. Luego, se busca el objeto minorista en self.next_agentes y se llama al método minorista.recibe_producto_del_mayorista para que el minorista reciba el producto. En caso de que el pedido esté marcado como 'NO_PROCESADA', se agrega a la lista llamada self.nuevas_ordenes_pendientes_MINO. Pero si la orden aún tiene días de entrega restantes (dias_tiempo_entrega mayor que 0) y está en estado 'PROCESADA', se decrementa el contador de días de entrega en 1 y se añade a la lista self.nuevas_ordenes_pendientes_MINO. Las órdenes que permanecen como 'NO PROCESADA' también se añaden a esta lista nuevas_ordenes_pendientes_MINO. Finalmente, la lista original de ordenes_pendientes_MINO se actualiza con la nueva lista nuevas_ordenes_pendientes_MINO, y se calcula el total de productos en pedidos no procesados (total_pedidos_pendientes_MINO en estado 'NO_PROCESADA'), imprimiendo el resultado al final. Todo lo anterior permite un seguimiento eficiente y preciso de las órdenes.

Un método (**calculo_cantidad_a_ordenar_al_productor)** que calcula la cantidad que el mayorista debe ordenar a los productores. Calcula el punto de reorden, considerando la incertidumbre en la demanda y el tiempo de entrega (utiliza un valor Z=1.96 para un nivel de confianza del 95%). Si la posición de inventario (self.posicion_inventario) es menor que el punto de reorden, se debe generar una orden a los productores para alcanzar el nivel objetivo.

Un método (**generar_orden_al_productor**) para se utiliza para generar órdenes de compra a los productores (P1 y P2). Verifica si hay cantidad a ordenar (cantidad_a_ordenar_mayo_p1 > 0), genera la orden (mayorista_orden_p1), actualiza la cantidad pendiente por recibir (self.cantidad_PENDIENTE_a_recibir_de_los_productores), y envía la orden de pedido a los productores para que sea procesada (P1.productor_recibe_orden_de_compra_del_mayorista). El mismo procedimiento realiza para cantidad_a_ordenar_mayo_p2.

Un método para realizar las actualizaciones necesarias cuando se recibe el producto desde los productores (**recibe_producto_del_productor**) y se encarga de actualizar el stock disponible en el mayorista (self.stock_producto_TERMINADO_mayorista). También ajusta la cantidad pendiente a recibir de los productores (self.cantidad_PENDIENTE_a_recibir_de_los_productores), dado que ahora parte o el pedido completo ha sido recibido.

***C3. Métodos de la clase productor*.** Los métodos de esta clase permiten gestionar la operación de un objeto productor en la cadena de suministro. El primero es el **constructor** (__init__) para inicializar los atributos del objeto cuando se crea una instancia de la clase. Configura parámetros como el nombre del productor, los datos sobre la media y desviación estándar de la demanda diaria (media_demanda, desv_demanda), los tiempos mínimo y máximos de entrega al mayorista (min_tiempo_entrega y max_tiempo_entrega), el tiempo mínimo y

máximo de producción (min_tiempo_pcc y min_tiempo_pcc), el stock objetivo de productos terminados que el productor intenta mantener (nivel_objetivo), el stock de producto en proceso (stock_producto_en_proceso_PRODU), el inventario de productos terminados (stock_producto_TERMINADO_productor), el límite máximo de producción (max_limite_pcc).

Adicionalmente, se definen otros atributos y listas como:

Unas variables para almacenar el inventario de material proveniente de cada proveedor (stock_material_S1, stock_material_S2, stock_material_S3),

Una variable que almacena la cantidad de la orden a producir según la capacidad de fabricación (cantidad_de_orden_de_produccion_PRODU).

Una lista que almacena el tiempo de producción restante para la orden de producción (tiempo_de_produccion).

Una lista para almacenar las órdenes de producción (ordenes_de_produccion).

Una lista para almacenar las ordenes al finalizar el ciclo de producción (nuevas_ordenes_de_produccion).

Un contador que rastrea el número de órdenes de producción creadas (numero_orden), inicializado en 1.

Una lista (ordenes_pendientes_MAYO) para almacenar todas las órdenes que el productor ha recibido del mayorista pero que aún no han sido completamente procesadas o entregadas. Una orden se considera procesada cuando el productor ha disminuido el inventario en la cantidad de la orden, pero los productos aún están en tránsito esperando a ser entregados, y una orden se considera entregada cuando ha llegado a su destino; es decir, el mayorista finalmente recibe los productos. Esto ocurre después de un período de tiempo, representado por dias_tiempo_entrega. El código controla este período, y cuando dias_tiempo_entrega llega a 0, la orden se entrega al mayorista.

Una lista (nuevas_ordenes_pendientes_MAYO) que se utiliza como lista temporal donde se colocan las órdenes que, después del ciclo de procesamiento todavía deben estar pendientes (ya sea porque no se han procesado o porque aún les quedan días de entrega. Esta lista reemplaza a la lista ordenes_pendientes_MAYO al finalizar el ciclo de procesamiento, para que en el próximo día se siga trabajando con las órdenes que aún están pendientes. Su función es realizar un seguimiento de las órdenes que deben mantenerse en la lista de ordenes_pendientes_MAYO para el siguiente ciclo de procesamiento.

La variable booleana (dia_sin_stock_produ) para rastrear si el productor tiene suficiente stock para procesar todas las órdenes del día. Inicializada como “False” al comienzo de iniciar la revisión diaria de pedidos pendientes. La cual cambia a “True” si al barrer los pedidos pendientes no hubo suficiente stock para cubrir la demanda.

La lista que almacena el historial de días sin stock (historial_dias_sin_stock_produ)

El contador de días sin stock (contador_dias_sin_stock) que cuenta los días en los que hubo falta de stock. Inicializado en 0.

El punto_de_reorden (punto_de_reorden), que indica el nivel de inventario en el que el productor debe reabastecerse (inicialmente establecido en 0)

Una variable (cantidad_disponible_PRODU)

La cantidad de material que el productor espera recibir de cada productor (stock_material_PENDIENTE_a_recibir_de_S1, stock_material_PENDIENTE_a_recibir_de_S2, stock_material_PENDIENTE_a_recibir_de_S3)

Una variable (dias_para_recibir) relacionada con el tiempo de entrega que tarda en llegar un pedido del proveedor respectivo, luego de despachado. Inicialmente establecida en 0, se actualiza usando el método calculo_tiempo_entrega() de la instancia proveedor.

La variable self.total_pedidos_pendientes_MAYO que almacena el total de productos que están pendientes de ser procesados por el productor, es decir, aquellos pedidos que están es estado NO_PROCESADA (aún no han sido despachados a los mayoristas).

Los demás métodos incluyen:

Un método (**calculo_tiempo_entrega)** para calcular el tiempo de entrega de un pedido, que genera un número aleatorio dentro del rango min_tiempo_entrega y max_tiempo_entrega.

. Un método (**calculo_tiempo_de_produccion**) que calcula un tiempo de producción aleatorio en base a los tiempos mínimo y máximo de producción (min_tiempo_pcc y max_tiempo_pcc).

Un método (**produccion_PRODU**) que gestiona el proceso de producción del productor el cual realiza varias operaciones:

- Se visualizan los inventarios actuales (stock de producto terminado y cantidades de materiales disponibles de los proveedores (S1, S2 y S3)).
- Se calcula el material_disponible como el mínimo entre los tres materiales
- Si material_disponible >= 1 and self.stock_producto_en_proceso_PRODU < self.max_limite_pcc, se calcula cuánta cantidad adicional se puede producir (cantidad_de_orden_de_produccion_PRODU)
- El stock de materiales de los proveedores se reduce en la cantidad correspondiente a la orden.
- El stock de productos en proceso se incrementa en la cantidad a producir.
- Se guarda la nueva orden en la lista ordenes_de_produccion, que contiene detalles como el número de orden, el tiempo de producción y la cantidad a producir.

- Se actualizan las órdenes de producción; iterando sobre las órdenes existentes. Si una orden ha completado su tiempo de producción, se transfiere la cantidad del producto en proceso al stock de producto terminado. Si no, se reduce en 1 el tiempo restante de la orden.
- Al final, se visualiza el stock de productos en proceso y el stock de productos terminados.

Un método (**productor_recibe_orden_de_compra_del_mayorista**) que se encarga de recibir la orden de pedido del mayorista y almacenarla en la lista de órdenes pendientes (ordenes_pendientes_MAYO) sin procesarla inmediatamente. El método recibe como parámetro un diccionario llamado orden_mayorista, que contiene claves que le fueron asignadas como el nombre del mayorista que envía la orden 'mayorista', la cantidad de productos que el mayorista solicita 'cantidad', los 'dias_tiempo_entrega', el tiempo estimado de entrega para cumplir la orden después de ser procesada y el nombre del productor a quien va dirigida 'productor'. Este método inicialmente extrae la información del diccionario y la asigna a variables locales. Posteriormente, el método almacena la orden en la lista ordenes_pendientes_MAYO en 'estado' de 'NO_PROCESADA'. Para ello almacena la orden utilizando las claves 'mayorista': que guarda el nombre del mayorista que hizo la solicitud, 'cantidad': que guarda la cantidad de productos solicitados, 'dias_tiempo_entrega': que almacena el tiempo estimado en días para que el productor después de procesada la orden la entregue, 'productor': que guarda el nombre del productor que recibió la orden de pedido, el 'estado': marcado como 'NO_PROCESADA'; lo que indica que aún no ha sido atendida, y los 'dias_espera': un contador inicializado en 0 que sirve para registrar cuántos días han pasado desde que la orden de compra fue recibida por el productor.

Un método (**productor_procesa_ordenes_pendientes_del_mayorista**) cuyo propósito es tratar de procesar todas las órdenes pendientes del mayorista, de acuerdo al inventario de productos terminados. Para comenzar, se inicializa la bandera (self. dia_sin_stock_produ) en “False”,

utilizada para verificar si durante el día, el productor se quedó sin stock suficiente para procesar las órdenes. Luego, para cada orden de la lista (ordenes_pendientes_MAYO) aumenta en uno el número de días que ha estado esperando cada orden (orden['dias_espera'] += 1). Verifica si la orden está en estado "NO_PROCESADA", y si el stock del productor (self.stock_producto_TERMINADO_productor) es suficiente para cubrir la cantidad completa de la orden, si sí, se reduce el inventario en la cantidad respectiva, y la orden se marca como "PROCESADA", pero si el stock del productor no es suficiente para cumplir con toda la cantidad solicitada y aún queda algo de stock, se procesa parcialmente la orden. En este caso: La cantidad que se puede despachar (cantidad_despachada = self.stock_producto_TERMINADO_productor) se resta del stock disponible, y se genera una nueva orden con la cantidad restante no procesada. Esta nueva orden generada se agrega a la lista de ordenes_pendientes_MAYO justo después de la orden actual. Se marca la orden actual como "PROCESADA" y se establece la bandera dia_sin_stock_produ en True para indicar que no se pudo satisfacer completamente. Pero, si el productor no tiene suficiente stock para despachar ninguna cantidad de la orden, se marca la bandera dia_sin_stock_produ como True y se continúa con la siguiente orden. Al finalizar la iteración, se guarda en el historial si el productor tuvo o no problemas de stock ese día, añadiendo el valor de dia_sin_stock_produ al historial de días sin stock (historial_dias_sin_stock_produ). El contador de días sin stock (contador_dias_sin_stock) también se incrementa en 1 si el productor experimentó al menos una orden no procesada por falta de stock durante el día (dia_sin_stock_produ es True).

Después, se recorre cada orden de la lista self. ordenes_pendientes_MAYO, verificando los días restantes para su entrega. Si una orden tiene dias_tiempo_entrega igual a 0 y está marcado como 'PROCESADA', se extrae la cantidad de producto a enviar y el nombre del mayorista

asociado. Luego, se busca el objeto mayorista en self.next_agentes y se llama al método mayorista.recibe_producto_del_productor para que el mayorista reciba el producto. En caso de que el pedido esté marcado como 'NO_PROCESADA', se agrega a la lista llamada self.nuevas_ordenes_pendientes_MAYO. Pero si la orden aún tiene días de entrega restantes (dias_tiempo_entrega mayor que 0) y está en estado 'PROCESADA', se decrementa el contador de días de entrega en 1 y se añade a la lista self.nuevas_ordenes_pendientes_MAYO. Las órdenes que permanecen como 'NO PROCESADA' también se añaden a esta lista self.nuevas_ordenes_pendientes_MAYO. Finalmente, la lista original de ordenes_pendientes_MAYO se actualiza con la nueva lista self.nuevas_ordenes_pendientes_MAYO, y se calcula el total de productos en pedidos no procesados (total_pedidos_pendientes_MAYO en estado 'NO_PROCESADA'), imprimiendo el resultado al final. Todo lo anterior permite un seguimiento eficiente y preciso de las órdenes.

Un método (**punto_de_reorden_PRODU**) que calcula el punto de reorden (ROP) del productor:

1.Se determina el tiempo de entrega promedio que tarda cada proveedor (S1, S2, S3) en entregar un pedido. El tiempo de entrega sigue una distribución uniforme (Ejm. *tiempo_entrega_promedio_S1 = (S1.min_tiempo_entrega + S1.max_tiempo_entrega) / 2*). Luego, se determina cuál es el mayor tiempo de entrega entre los tres proveedores, para calcular el cuello de botella:

tiempo_de_entrega_promedio=max(tiempo_entrega_promedio_S1,

tiempo_entrega_promedio_S2, tiempo_entrega_promedio_S3)

2.Se determina la desviación estándar del tiempo de entrega (σ_LT): También para cada proveedor, se calcula la desviación estándar del tiempo de entrega. (Ejm. *sigma_LT_S1 =*

*(S1.max_tiempo_entrega - S1.min_tiempo_entrega) / (12 ** 0.5)*). Luego, se selecciona el mayor de los tres valores para el cálculo del cuello de botella:

sigma_LT = max(sigma_LT_S1, sigma_LT_S2, sigma_LT_S3)

3.Se determina el tiempo de procesamiento promedio (tiempo medio que tarda el productor en procesar los materiales y convertirlos en producto terminado). El tiempo de procesamiento sigue una distribución uniforme:

tiempo_procesamiento_material_promedio = (self.min_tiempo_pcc + self.max_tiempo_pcc) / 2

4.Se determina la desviación estándar del tiempo de procesamiento:

*sigma_procesamiento = (self.max_tiempo_pcc - self.min_tiempo_pcc) / (12 ** 0.5)*

5.Finalmente se calcula el punto de reorden (ROP), usando la demanda media, la desviación estándar combinada, y el factor de seguridad para un nivel de confianza del 95% (Z=1.96):

*self.punto_de_reorden = int((self.media_demanda * tiempo_total) + Z * sigma_total)*

Unos métodos (**calculo_cantidad_a_ordenar_al_proveedorS1**), (**calculo_cantidad_a_ordenar_al_proveedorS2**) y (**calculo_cantidad_a_ordenar_al_proveedorS3**) para calcular cuánto material necesita ordenar el productor a cada uno de sus proveedores (S1, S2, S3). Se sigue el siguiente proceso: Se determina la posición de inventario (posicion_inventario) y se verifica si la cantidad es menor o igual al punto de reorden, se calcula cuánto material ordenar para alcanzar el nivel objetivo. Si no es necesario ordenar más material, la cantidad es 0.

Un método (**generar_orden_al_proveedor**) que genera una orden de compra para cada proveedor (S1, S2, S3) dependiendo de las cantidades calculadas en los métodos anteriores (calculo_cantidad_a_ordenar_al_proveedorS1, calculo_cantidad_a_ordenar_al_proveedorS2,

calculo_cantidad_a_ordenar_al_proveedorS3). Para cada proveedor, si la cantidad a ordenar es mayor que 0, se establece el tiempo de entrega y se crea un diccionario con los detalles de la orden, luego se llama al método en el objeto proveedor (Ejm: S1.proveedor_recibe_orden_de_compra_del_productor) para notificarle la orden. Finalmente, se actualiza el stock de materiales pendientes que el productor espera recibir del proveedor (Ejm: self.stock_material_PENDIENTE_a_recibir_de_S1 += cantidad_a_ordenar_produc_s1) .

Por último, se tiene un método (**productor_recibe_materiales_del_proveedor**) que actualiza el inventario del productor cuando recibe materiales de un proveedor. Dependiendo del proveedor (S1, S2 o S3), la cantidad recibida se suma al stock correspondiente del productor y se imprime el estado del inventario después de la recepción. Además, se ajusta la cantidad pendiente de recibir de ese proveedor, restando la cantidad ya entregada, lo que permite mantener un seguimiento preciso del inventario disponible y pendiente.

C4. Métodos de la clase proveedor. Los métodos de esta clase modelan el comportamiento de un objeto proveedor en la cadena de suministro**.** El primero es el **constructor** (__init__) que en el caso de la clase proveedor, configura parámetros como el nombre del proveedor, los datos sobre la media y desviación estándar de la demanda diaria (media_demanda, desv_demanda), los tiempos mínimo y máximos de entrega al productor (min_tiempo_entrega y max_tiempo_entrega), el tiempo mínimo y máximo de producción (min_tiempo_pcc y max_tiempo_pcc), el stock objetivo de material procesado que el proveedor intenta mantener (nivel_objetivo), el stock de material en proceso (stock_material_en_proceso), el inventario de productos terminados listos para ser despachados (stock_material_provee_PROCESADO), el límite máximo de producción (max_limite_pcc).

Adicionalmente, se definen otros atributos y listas como:

Unas variables para almacenar el inventario de material (stock_material_proveedor),

Una variable que almacena la cantidad de la orden a producir según la capacidad de fabricación (cantidad_de_orden_de_produccion_PROVEE).

Una lista que almacena el tiempo de producción restante para la orden de producción (tiempo_de_produccion).

Una lista para almacenar las órdenes de producción (ordenes_de_produccion).

Una lista para almacenar las ordenes al finalizar el ciclo de producción (nuevas_ordenes_de_produccion).

Un contador que rastrea el número de órdenes de producción creadas (numero_orden), inicializado en 1.

Una lista (ordenes_pendientes_PRODU) para almacenar todas las órdenes que el proveedor ha recibido del productor pero que aún no han sido completamente procesadas o entregadas. Una orden se considera procesada cuando el proveedor ha disminuido el inventario en la cantidad de la orden, pero los productos aún están en tránsito esperando a ser entregados, y una orden se considera entregada cuando ha llegado a su destino; es decir, el productor finalmente recibe los productos. Esto ocurre después de un período de tiempo, representado por dias_tiempo_entrega. El código controla este período, y cuando dias_tiempo_entrega llega a 0, la orden se entrega al productor.

Una lista (nuevas_ordenes_pendientes_PRODU) que se utiliza como lista temporal donde se colocan las órdenes que, después del ciclo de procesamiento todavía deben estar pendientes (ya sea porque no se han procesado o porque aún les quedan días de entrega. Esta lista reemplaza a la lista ordenes_pendientes_PRODU al finalizar el ciclo de procesamiento, para que en el próximo día se siga trabajando con las órdenes que aún están pendientes. Su función es realizar un

seguimiento de las órdenes que deben mantenerse en la lista de ordenes_pendientes_PRODU para el siguiente ciclo de procesamiento.

La variable booleana (dia_sin_stock_provee) para rastrear si el proveedor tiene suficiente stock para procesar todas las órdenes del día. Inicializada como “False” al comienzo de iniciar la revisión diaria de pedidos pendientes. La cual cambia a “True” si al barrer los pedidos pendientes no hubo suficiente stock para cubrir la demanda.

La lista que almacena el historial de días sin stock (historial_dias_sin_stock_provee)

El contador de días sin stock (contador_dias_sin_stock) que cuenta los días en los que hubo falta de stock. Inicializado en 0.

El punto_de_reorden (punto_de_reorden), que indica el nivel de inventario en el que el proveedor debe reabastecerse (inicialmente establecido en 0)

Una variable (cantidad_disponible_PROVEE)

La cantidad de material que el proveedor espera obtener (cantidad_PENDIENTE)

Una variable (dias_para_obtener_PROVEE) relacionada con el tiempo de entrega que tarda en el proveedor en obtener un pedido. Establecido en 0.

Una variable (orden_pendiente_PROVEE)

La variable self.total_pedidos_pendientes_PRODU que almacena el total de productos que están pendientes de ser procesados por el proveedor, es decir, aquellos pedidos que están es estado NO_PROCESADA (aún no han sido despachados a los productores).

Los demás métodos incluyen:

Un método (**calculo_tiempo_entrega)** para calcular el tiempo de entrega de un pedido, que genera un número aleatorio dentro del rango min_tiempo_entrega y max_tiempo_entrega.

. Un método (**calculo_tiempo_de_produccion**) que calcula un tiempo de producción aleatorio en base a los tiempos mínimo y máximo de producción (min_tiempo_pcc y max_tiempo_pcc).

Un método (**produccion_PROVEE**) que lleva a cabo el proceso de producción del proveedor realizando varias operaciones:

- Se visualiza el inventario actual (stock_material_proveedor, stock_material_en_proceso y stock_material_provee_PROCESADO).
- Si hay material (stock_material_proveedor) y el material en proceso (stock_material_en_proceso) es menor al límite de producción (max_limite_pcc) se calcula la cantidad de material que se puede procesar.
  self.cantidad_de_orden_de_produccion_PROVEE = min(self.stock_material_proveedor, (self.max_limite_pcc - self.stock_material_en_proceso))
- Se resta la cantidad de la orden de producción del material disponible (self.stock_material_proveedor -= self.cantidad_de_orden_de_produccion_PROVEE) y se suma la cantidad de la orden al material en proceso (self.stock_material_en_proceso += self.cantidad_de_orden_de_produccion_PROVEE)
- Se estima el tiempo de producción de la orden (self.calculo_tiempo_de_produccion)
- Se almacena la nueva orden en la lista ordenes_de_produccion, que contiene detalles como el número de la orden, el tiempo de producción y la cantidad a producir.
- Se incrementa el número de orden en 1
- Se actualizan las órdenes; iterando sobre las órdenes de producción existentes. Si una orden ha completado su tiempo de producción, se transfiere la cantidad del producto en proceso al stock de producto terminado. Si no, se reduce en 1 el tiempo restante de la orden.
- Al final, se visualiza el stock de productos en proceso y el stock de productos terminados.

Un método (**proveedor_recibe_orden_de_compra_del_productor**) que se encarga de recibir la orden de pedido del productor y almacenarla en la lista de órdenes pendientes (ordenes_pendientes_PRODU) sin procesarla inmediatamente. El método recibe como parámetro el diccionario llamado orden_productor, que contiene claves que le fueron asignadas como el nombre del productor que envía la orden 'productor', la cantidad de productos que el productor solicita 'cantidad', los 'dias_tiempo_entrega', el tiempo estimado de entrega para cumplir la orden después de ser procesada y el nombre del proveedor a quien va dirigida 'proveedor'. Este método inicialmente extrae la información del diccionario y la asigna a variables locales. Posteriormente, el método almacena la orden en la lista ordenes_pendientes_PRODU en 'estado' de 'NO_PROCESADA'. Para almacenarla con las claves 'productor': que guarda el nombre del productor que hizo la solicitud, 'cantidad': que guarda la cantidad de productos solicitados, 'dias_tiempo_entrega': que almacena el tiempo estimado en días para que el proveedor después de procesada la orden la entregue, 'proveedor': que guarda el nombre del proveedor que recibió la orden de pedido, el 'estado': marcado como 'NO_PROCESADA'; lo que indica que aún no ha sido atendida, y los 'dias_espera': un contador inicializado en 0 que sirve para registrar cuántos días han pasado desde que la orden de compra fue recibida por el proveedor.

Un método (**proveedor_procesa_ordenes_pendientes_del_productor**) cuyo propósito es tratar de procesar todas las órdenes pendientes del productor, de acuerdo al inventario de material procesado. Para comenzar, se inicializa la bandera (self. dia_sin_stock_provee) en “False”, utilizada para verificar si durante el día, el proveedor se quedó sin stock suficiente para procesar las órdenes. Luego, para cada orden de la lista (ordenes_pendientes_PRODU) aumenta en uno el número de días que ha estado esperando cada orden (orden['dias_espera'] += 1). Verifica si la orden está en estado "NO_PROCESADA", y si el stock del proveedor (self.

stock_material_provee_PROCESADO) es suficiente para cubrir la cantidad completa de la orden, si sí, se reduce el inventario en la cantidad respectiva, y la orden se marca como "PROCESADA", pero si el stock del proveedor no es suficiente para cumplir con toda la cantidad solicitada y aún queda algo de stock, se procesa parcialmente la orden. En este caso: La cantidad que se puede despachar (cantidad_despachada = self.stock_material_provee_PROCESADO) se resta del stock disponible, y se genera una nueva orden con la cantidad restante no procesada. Esta nueva orden generada se agrega a la lista de ordenes_pendientes_PRODU justo después de la orden actual. Se marca la orden actual como "PROCESADA" y se establece la bandera dia_sin_stock_provee en True para indicar que no se pudo satisfacer completamente. Pero, si el proveedor no tiene suficiente stock para despachar ninguna cantidad de la orden, se marca la bandera dia_sin_stock_provee como True y se continúa con la siguiente orden. Al finalizar la iteración, se guarda en el historial si el proveedor tuvo o no problemas de stock ese día, añadiendo el valor de dia_sin_stock_provee al historial de días sin stock (historial_dias_sin_stock_provee). El contador de días sin stock (contador_dias_sin_stock) también se incrementa en 1 si el proveedor experimentó al menos una orden no procesada por falta de stock durante el día (dia_sin_stock_provee es True).

Después, se recorre cada orden de la lista self.ordenes_pendientes_PRODU, verificando los días restantes para su entrega. Si una orden tiene dias_tiempo_entrega igual a 0 y está marcado como 'PROCESADA', se extrae la cantidad de producto a enviar y el nombre del productor asociado. Luego, se busca el objeto productor en self.next_agentes y se llama al método productor.productor_recibe_materiales_del_proveedor para que el productor reciba el producto. En caso de que el pedido esté marcado como 'NO_PROCESADA', se agrega a la lista llamada self.nuevas_ordenes_pendientes_PRODU. Pero si la orden aún tiene días de entrega restantes (dias_tiempo_entrega mayor que 0) y está en estado 'PROCESADA', se decrementa el contador

de días de entrega en 1 y se añade a la lista self. nuevas_ordenes_pendientes_PRODU. Las órdenes que permanecen como 'NO PROCESADA' también se añaden a esta lista self. nuevas_ordenes_pendientes_PRODU. Finalmente, la lista original de ordenes_pendientes_PRODU se actualiza con la nueva lista self. nuevas_ordenes_pendientes_PRODU, y se calcula el total de productos en pedidos no procesados (total_pedidos_pendientes_PRODU en estado 'NO_PROCESADA'), imprimiendo el resultado al final. Todo lo anterior permite un seguimiento eficiente y preciso de las órdenes.

Un método (**calculo_cantidad_necesaria_mp**) que calcula el punto de reorden (ROP) del proveedor y la cantidad de materia prima necesaria:

1. Se determina el tiempo que el proveedor tarda en obtener la materia prima (*calculo_tiempo_obtencionmp_Provee = self.dias_para_obtener_PROVEE)* el cual es *0* porque los *dias_para_obtener_PROVEE* se establecieron en 0, y por lo tanto no hay incertidumbre, luego *sigma_LT = 0*

2. Se determina el tiempo de procesamiento promedio (tiempo medio que tarda el proveedor en procesar el material y convertirlo en material procesado). El tiempo de procesamiento sigue una distribución uniforme:

tiempo_procesamiento_material_prom_provee = (self.min_tiempo_pcc + self.max_tiempo_pcc) / 2

3. Se determina la desviación estándar del tiempo de procesamiento:

*sigma_procesamiento = (self.max_tiempo_pcc - self.min_tiempo_pcc) / (12 ** 0.5)*

4. Se calcula el punto de reorden (ROP), usando la demanda media, la desviación estándar combinada, y el factor de seguridad para un nivel de confianza del 95% (Z=1.96):

*self.punto_de_reorden = int((self.media_demanda * tiempo_total) + Z * sigma_total)*

5. Se calcula a calcular cuánto material necesita el proveedor. Se sigue el siguiente proceso: Se determina la posición de inventario (posicion_inventario) y se verifica si la cantidad es menor o igual al punto de reorden, se calcula cuánto material ordenar para alcanzar el nivel objetivo. Si no es necesario ordenar más material, la cantidad es 0.

Un método (**generar_orden_mp**) que genera una orden de necesidad de materia prima dependiendo de la cantidad del método anterior (cantidad_necesaria_mp_provee). Si la cantidad necesaria es mayor que 0, se establecen los días para obtenerla (dias_para_obtener_PROVEE), se crea un diccionario (orden_provee) con los detalles de la orden, se agrega a las ordenes pendientes necesarias del proveedor, se actualiza la cantidad pendiente (self.cantidad_PENDIENTE). Para recibir esta orden u obtener esta materia prima, se ejecuta el método (procesar_orden_mp) de la misma instancia proveedor.

El método **(procesar_orden_mp)** itera sobre las órdenes pendientes de proveedores almacenadas en self.orden_pendiente_PROVEE, extrae la cantidad y el nombre del proveedor de cada orden, luego busca en la lista de proveedores (lista_proveedores) la instancia del proveedor correspondiente (proveedor_instancia.nombre) y llama a su método (proveedor_instancia.provee_recibe_material) para notificarle la recepción del material. Una vez procesadas todas las órdenes, la lista de órdenes pendientes (orden_pendiente_PROVEE) se vacía.

Por último, se tiene el método (**provee_recibe_material**) que actualiza el inventario del proveedor cuando obtiene la materia prima necesaria. Se suma al stock correspondiente del proveedor (self.stock_material_proveedor += cantidad_producto_enviadoProveeF) y se imprime el estado del inventario después de la recepción. Además, se ajusta la cantidad pendiente a obtener, restando la cantidad ya ingresada (self.cantidad_PENDIENTE -= cantidad_producto_enviadoProveeF).

Apéndice D. Código de Programación - Modelo de Red Bayesiana con Análisis de Riesgos y Resiliencia de la Cadena de Suministro

En el entorno de desarrollo de Google Colab, utilizando Python, el código es el siguiente:

```
import random
import math
import numpy as np
import pandas as pd

# Para negrita
negrita = "\033[1m"
fin_negrita = "\033[0m"

# Parámetros de entrada

# tiempos mínimos de producción para los diferentes actores de la cadena.
min_tiempo_pcc = {
    "S1": 6, "S2": 6, "S3": 6, "P1": 4, "P2": 4, "W1": 0, "W2": 0, "R1":
0, "R2": 0, "R3": 0   #Tiempo mínimo de producción --> Uniforme
}

# tiempos máximos de producción para los diferentes actores de la cadena.
max_tiempo_pcc = {
    "S1": 10, "S2": 10, "S3": 10, "P1": 6, "P2": 6, "W1": 0, "W2": 0,
"R1": 0, "R2": 0, "R3": 0  #Tiempo maximo de producción --> Uniforme
}

# tiempos mínimos de entrega para los diferentes actores de la cadena.
min_tiempo_entrega = {
    "S1": 4, "S2": 4, "S3": 4, "P1": 3, "P2": 3, "W1": 1, "W2": 1, "R1":
0, "R2": 0, "R3": 0   #Los actores demoran mínimo x días en entregar el
pedido desde que es despachado
}

# tiempos máximos de entrega para los diferentes actores de la cadena.
max_tiempo_entrega = {
    "S1": 8, "S2": 8, "S3": 8, "P1": 5, "P2": 5, "W1": 3, "W2": 3, "R1":
0, "R2": 0, "R3": 0   #Los actores demoran máximo x días en entregar el
pedido desde que es despachado
}

#nivel de stock objetivo
```

```
nivel_objetivo = {
    "S1": 11000, "S2": 11000, "S3": 12000, "P1": 3000, "P2": 4000, "W1":
2000, "W2": 4000, "R1": 1500, "R2": 1500, "R3": 1500
}

# Realizar réplicas de la simulación
num_replicas = 5
num_dias = 1800

# Almacenar las matrices de probabilidades condicionales para cada réplica
matrices_probabilidades_replicas_S1 = []
matrices_probabilidades_replicas_S2 = []
matrices_probabilidades_replicas_S3 = []
matrices_probabilidades_replicas_P1 = []
matrices_probabilidades_replicas_P2 = []
matrices_probabilidades_replicas_W1 = []
matrices_probabilidades_replicas_W2 = []
matrices_probabilidades_replicas_R1 = []
matrices_probabilidades_replicas_R2 = []
matrices_probabilidades_replicas_R3 = []

for replica in range(num_replicas):

    class proveedor:
        def __init__(self, nombre, media_demanda, desv_demanda,
min_tiempo_entrega, max_tiempo_entrega, min_tiempo_pcc, max_tiempo_pcc,
nivel_objetivo, stock_material_en_proceso,
stock_material_provee_PROCESADO, max_limite_pcc):
            self.nombre = nombre
            self.media_demanda = media_demanda
            self.desv_demanda = desv_demanda
            self.min_tiempo_entrega = min_tiempo_entrega
            self.max_tiempo_entrega = max_tiempo_entrega
            self.min_tiempo_pcc = min_tiempo_pcc
            self.max_tiempo_pcc = max_tiempo_pcc
            self.nivel_objetivo = nivel_objetivo

            self.stock_material_en_proceso = stock_material_en_proceso
            self.stock_material_provee_PROCESADO =
stock_material_provee_PROCESADO
            self.max_limite_pcc = max_limite_pcc

            self.stock_material_proveedor = 0

            self.cantidad_de_orden_de_produccion_PROVEE = 0
```

```
            self.tiempo_de_produccion = []
            self.ordenes_de_produccion = []  # Inicializa una lista para
almacenar las órdenes de producción
            self.nuevas_ordenes_de_produccion = []
            self.numero_orden = 1  # Inicializa el número de orden en 1

            self.ordenes_pendientes_PRODU = []  # Lista para almacenar las
órdenes de compra pendientes que ha generado el productor
            self.nuevas_ordenes_pendientes_PRODU = []

            self.dia_sin_stock_provee = False  # Inicializar la bandera
dia_sin_stock_provee
            self.historial_dias_sin_stock_provee = []
            self.contador_dias_sin_stock = 0

            self.punto_de_reorden = 0
            self.cantidad_disponible_PROVEE = 0
            self.cantidad_PENDIENTE = 0
            self.dias_para_obtener_PROVEE = 0
            self.orden_pendiente_PROVEE = [] # Lista para almacenar la
órden de materia prima que necesita el proveedor

            self.total_pedidos_pendientes_PRODU = 0

        def calculo_tiempo_entrega(self):
            return random.randint(self.min_tiempo_entrega,
self.max_tiempo_entrega)

        def calculo_tiempo_de_produccion(self):
            return random.randint(self.min_tiempo_pcc,
self.max_tiempo_pcc)

        def produccion_PROVEE(self):
            print("stock MATERIAL proveedor INICIO DIA:",
self.stock_material_proveedor)
            print("stock material en PROCESO INICIO DIA:",
self.stock_material_en_proceso)
            print(f"{negrita}stock material proveedor PROCESADO INICIO
DIA:{fin_negrita}", self.stock_material_provee_PROCESADO)

            if self.stock_material_proveedor >= 1 and
self.stock_material_en_proceso < self.max_limite_pcc:

                # Calcular la cantidad de material adicional que se puede
procesar (capacidad_para_nuevas_ordenes)
```

```
                self.cantidad_de_orden_de_produccion_PROVEE =
min(self.stock_material_proveedor, (self.max_limite_pcc -
self.stock_material_en_proceso))
                print("cantidad adicional a PRODUCIR:",
self.cantidad_de_orden_de_produccion_PROVEE)

                # Restar la cantidad de la orden de producción del
material disponible
                self.stock_material_proveedor -=
self.cantidad_de_orden_de_produccion_PROVEE
                print("stock material proveedor DESPUES DE PRODUCCIÓN:",
self.stock_material_proveedor)

                # Sumar la cantidad de la orden de producción al material
en proceso
                self.stock_material_en_proceso +=
self.cantidad_de_orden_de_produccion_PROVEE
                print("stock material en PROCESO CON HOY:",
self.stock_material_en_proceso)

                # Calcular el tiempo de producción
                self.tiempo_de_produccion =
self.calculo_tiempo_de_produccion()
                print("tiempo de produccion provee:",
self.tiempo_de_produccion)

                # Almacenar la orden de producción en la lista con el
número de la orden, tiempo de producción y cantidad de la orden de
producción
                self.ordenes_de_produccion.append({
                    'numero_orden': self.numero_orden,
                    'tiempo_de_produccion': self.tiempo_de_produccion,
                    'cantidad_orden':
self.cantidad_de_orden_de_produccion_PROVEE
                })

                self.numero_orden += 1  # Incrementa el número de orden

            # Iterar sobre las órdenes de producción existentes y
actualizar el tiempo restante de producción
            self.nuevas_ordenes_de_produccion = []
            for orden in self.ordenes_de_produccion:
                tiempo_restante = orden['tiempo_de_produccion']
                if tiempo_restante > 1:
                    tiempo_restante -= 1
```

```
                    orden['tiempo_de_produccion'] = tiempo_restante
                    self.nuevas_ordenes_de_produccion.append(orden)
                else:
                    self.stock_material_en_proceso -=
orden['cantidad_orden']
                    self.stock_material_provee_PROCESADO +=
orden['cantidad_orden']
            print(f"{negrita}stock material en PROCESO FIN DE
DIA:{fin_negrita}", self.stock_material_en_proceso)
            print(f"{negrita}stock material proveedor PROCESADO FIN DE
DIA:{fin_negrita}", self.stock_material_provee_PROCESADO)
            self.ordenes_de_produccion = self.nuevas_ordenes_de_produccion

            if not self.ordenes_de_produccion:
                print("No hay órdenes de producción en curso.")

        # La orden de pedido que le coloca el productor al proveedor

        def proveedor_recibe_orden_de_compra_del_productor(self,
nombre_productor, orden_productor):

            nombre_productor_recibido_por_proveedor =
orden_productor['productor']
            cantidad_ordenada_produ_recibido_por_provee =
orden_productor['cantidad']
            dias_tiempo_entrega_provee =
orden_productor['dias_tiempo_entrega']
            nombre_proveedor_recibido_por_provee =
orden_productor['proveedor']

            # Almacena la orden en la lista de órdenes pendientes, en
estado NO_PROCESADA
            self.ordenes_pendientes_PRODU.append({
                'productor': nombre_productor,
                'cantidad': cantidad_ordenada_produ_recibido_por_provee,
                'dias_tiempo_entrega': dias_tiempo_entrega_provee,
                'proveedor': nombre_proveedor_recibido_por_provee,
                'estado': 'NO_PROCESADA',
                'dias_espera': 0
            })

        def proveedor_procesa_ordenes_pendientes_del_productor(self):
            self.dia_sin_stock_provee = False    # Inicializar la bandera
            self.nuevas_ordenes_pendientes_PRODU = []
```

```
            for i, orden in enumerate(self.ordenes_pendientes_PRODU):
                orden['dias_espera'] += 1  # Incrementa los días de espera
en 1
                if orden['estado'] == 'NO_PROCESADA':
                    if orden['cantidad'] <=
self.stock_material_provee_PROCESADO:
                        self.stock_material_provee_PROCESADO -=
orden['cantidad']
                        orden['estado'] = 'PROCESADA'  # Cambio
NO_PROCESADA por PROCESADA
                    elif self.stock_material_provee_PROCESADO > 0:
                        # Despachar parcialmente
                        cantidad_despachada =
self.stock_material_provee_PROCESADO
                        orden_restante = {
                            'productor': orden['productor'],
                            'cantidad': orden['cantidad'] -
cantidad_despachada,
                            'dias_tiempo_entrega':
orden['dias_tiempo_entrega'],
                            'proveedor': orden['proveedor'],
                            'estado': 'NO_PROCESADA',
                            'dias_espera': orden['dias_espera']
                        }
                        # Actualizar la orden actual con la cantidad
despachada
                        orden['cantidad'] = cantidad_despachada
                        orden['estado'] = 'PROCESADA'
                        self.stock_material_provee_PROCESADO -=
cantidad_despachada
                        # Insertar la orden restante después de la actual
                        self.ordenes_pendientes_PRODU.insert(i + 1,
orden_restante)

                        # Actualizar la bandera
                        self.dia_sin_stock_provee = True
                    else:
                        # No hay suficiente stock para procesar la orden
                        self.dia_sin_stock_provee = True
                        continue

            self.historial_dias_sin_stock_provee.append(self.dia_sin_stock
_provee)
            #print(f"historial_dias_sin_stock de {self.nombre}:",
self.historial_dias_sin_stock_provee)
```

```
            # Incrementar el contador de días sin stock si hubo al menos
una venta perdida en el día
            if self.dia_sin_stock_provee:
                self.contador_dias_sin_stock += 1

            # Mostrar el contador actualizado
            print(f"contador_dias_sin_stock de {self.nombre}:",
self.contador_dias_sin_stock)

            for orden in self.ordenes_pendientes_PRODU:
                if orden['dias_tiempo_entrega'] == 0:
                    if orden['estado'] == 'PROCESADA':
                        cantidad_producto_enviadoProduF =
orden['cantidad']
                        nombre_productor = orden['productor']  # Obtener
el nombre del productor directamente del pedido
                        productor = self.next_agentes[nombre_productor]
                        productor.productor_recibe_materiales_del_proveedo
r(self.nombre, cantidad_producto_enviadoProduF)
                    elif orden['estado'] == 'NO_PROCESADA':
                          self.nuevas_ordenes_pendientes_PRODU.append(orde
n)  # Agregar el pedido a la nueva lista de pedidos pendientes
                elif orden['dias_tiempo_entrega'] > 0:
                    if orden['estado'] == 'PROCESADA':
                        orden['dias_tiempo_entrega'] -= 1  # Restar 1 a
los días de entrega
                        self.nuevas_ordenes_pendientes_PRODU.append(orden)
                    if orden['estado'] == 'NO_PROCESADA':
                        self.nuevas_ordenes_pendientes_PRODU.append(orden)

            # Reemplazar la lista original de pedidos pendientes con la
nueva lista
            self.ordenes_pendientes_PRODU =
self.nuevas_ordenes_pendientes_PRODU

            self.total_pedidos_pendientes_PRODU = sum(orden['cantidad']
for orden in self.ordenes_pendientes_PRODU if orden['estado'] ==
'NO_PROCESADA')
            print(f"Total pedidos pendientes NO PROCESADOS del productor:
{self.total_pedidos_pendientes_PRODU}")

        #Lo que realiza el proveedor para tener su materia prima.

        def calculo_cantidad_necesaria_mp(self):
```

```
            self.dias_para_obtener_PROVEE = 0     # El tiempo se
estableció en 0
            calculo_tiempo_obtencionmp_Provee =
self.dias_para_obtener_PROVEE
            print("tiempo para obtener el material provee:",
calculo_tiempo_obtencionmp_Provee)
            sigma_LT = 0      # No hay incertidumbre porque el tiempo se
estableció en 0.
            print("Desviación estándar del tiempo de entrega:", sigma_LT)

            tiempo_procesamiento_material_prom_provee =
(self.min_tiempo_pcc + self.max_tiempo_pcc) / 2    #tiempo promedio de
procesamiento (distribución uniforme)
            print("tiempo de procesamiento de material:",
tiempo_procesamiento_material_prom_provee)
            sigma_procesamiento = (self.max_tiempo_pcc -
self.min_tiempo_pcc) / (12 ** 0.5)  # Desviación estándar del tiempo de
procesamiento (distribución uniforme)
            print("desviación estándar del tiempo de procesamiento:",
sigma_procesamiento)
            tiempo_total = calculo_tiempo_obtencionmp_Provee +
tiempo_procesamiento_material_prom_provee
            print("Tiempo total:", tiempo_total)

            # ROP ajustado para incluir incertidumbre tanto en la demanda
como en el tiempo de entrega y procesamiento (su cálculo se simplifica
porque no hay incertidumbre en el tiempo de entrega)
            # Combinando la incertidumbre en la demanda y en el tiempo de
procesamiento. Desviación estándar total considerando demanda normal y
tiempo de procesamiento uniforme
            sigma_total = ((tiempo_total * self.desv_demanda ** 2) +
(self.media_demanda ** 2 * sigma_procesamiento ** 2)) ** 0.5
            print("Desviación estándar total:", sigma_total)
            Z = 1.96  # Valor Z para un nivel de confianza del 95%

            self.punto_de_reorden = int((self.media_demanda *
tiempo_total) + (Z * sigma_total))
            print("punto de reorden provee:", self.punto_de_reorden)

            print("stock material proveedor PROCESADO:",
self.stock_material_provee_PROCESADO)
            print("stock material en PROCESO proveedor:",
self.stock_material_en_proceso)
            print("stock MATERIAL proveedor:",
self.stock_material_proveedor)
```

```
            print("MATERIAL PENDIENTE a recibir:",
self.cantidad_PENDIENTE)
            print("Total pedidos pendientes NO PROCESADOS del productor:",
self.total_pedidos_pendientes_PRODU)

            self.posicion_inventario  =
self.stock_material_provee_PROCESADO + self.stock_material_en_proceso +
self.stock_material_proveedor + self.cantidad_PENDIENTE -
self.total_pedidos_pendientes_PRODU
            print(f"{negrita}posicion_inventario{fin_negrita}",
self.posicion_inventario)
            # Cálculo de la cantidad necesaria de materia prima
            if self.posicion_inventario  <= self.punto_de_reorden:
                self.cantidad_necesaria_mp_provee = self.nivel_objetivo -
self.posicion_inventario
            else:
                self.cantidad_necesaria_mp_provee = 0

            return self.cantidad_necesaria_mp_provee

        def generar_orden_mp(self, cantidad_necesaria_mp_provee):
            orden_provee = {}

            if cantidad_necesaria_mp_provee > 0:
                self.dias_para_obtener_PROVEE = 0
                # Almacenamiento de la información del pedido en el
diccionario orden_provee
                orden_provee = {
                    'proveedor': self.nombre,                    # A la
clave 'proveedor' se le asigna el valor self.nombre
                    'cantidad': cantidad_necesaria_mp_provee,    # A la
clave 'cantidad' se le asigna el valor cantidad_necesaria_mp_provee
                    'dias_tiempo_entrega':
self.dias_para_obtener_PROVEE,    # A la clave 'dias_tiempo_entrega' se le
asigna el valor dias_para_obtener_PROVEE
                    'dias_espera': 0
                }
                # Agrega orden pendiente a la lista de orden pendiente
                self.orden_pendiente_PROVEE.append(orden_provee)
                print("orden pendiente necesaria del proveedor:",
self.orden_pendiente_PROVEE)

                # Actualiza la cantidad pendiente
                self.cantidad_PENDIENTE += cantidad_necesaria_mp_provee
```

```
                print(f"cantidad PENDIENTE de {self.nombre} a recibir:",
self.cantidad_PENDIENTE)

        def procesar_orden_mp(self):

            for orden_provee in self.orden_pendiente_PROVEE:
                    cantidad_producto_enviadoProveeF =
orden_provee['cantidad']
                    nombre_proveedor_provee = orden_provee['proveedor']  #
Obtener el nombre del proveedor directamente del pedido
                    # Buscar la instancia correcta del proveedor en la
lista_proveedores
                    for proveedor_instancia in lista_proveedores:
                        if proveedor_instancia.nombre ==
nombre_proveedor_provee:
                            proveedor_instancia.provee_recibe_material(sel
f.nombre, cantidad_producto_enviadoProveeF)
                            break  # Termina el bucle una vez que se
encuentra el proveedor correcto

            self.orden_pendiente_PROVEE = []  # Limpiar la lista de
órdenes pendientes originales

        def provee_recibe_material(self, nombre_proveedor,
cantidad_producto_enviadoProveeF):
            print("proveedor que recibe", self.nombre)  # Accede al nombre
del proveedor desde self
            print("cantidad recibida:", cantidad_producto_enviadoProveeF)
            # Ingresar al stock el material recibido
            self.stock_material_proveedor +=
cantidad_producto_enviadoProveeF
            print(f"{negrita}stock MATERIAL proveedor DESPUES DE RECIBIR
la orden:{fin_negrita}", self.stock_material_proveedor)
            # Actualiza la cantidad pendiente
            self.cantidad_PENDIENTE -= cantidad_producto_enviadoProveeF
            print("cantidad PENDIENTE a recibir:",
self.cantidad_PENDIENTE)

    class productor:
        def __init__(self, nombre, media_demanda, desv_demanda,
min_tiempo_entrega, max_tiempo_entrega, min_tiempo_pcc, max_tiempo_pcc,
nivel_objetivo, stock_producto_en_proceso_PRODU,
stock_producto_TERMINADO_productor, max_limite_pcc):
            self.nombre = nombre
            self.media_demanda = media_demanda
```

```
            self.desv_demanda = desv_demanda
            self.min_tiempo_entrega = min_tiempo_entrega
            self.max_tiempo_entrega = max_tiempo_entrega
            self.min_tiempo_pcc = min_tiempo_pcc
            self.max_tiempo_pcc = max_tiempo_pcc
            self.nivel_objetivo = nivel_objetivo

            self.stock_producto_en_proceso_PRODU =
stock_producto_en_proceso_PRODU
            self.stock_producto_TERMINADO_productor =
stock_producto_TERMINADO_productor
            self.max_limite_pcc = max_limite_pcc

            self.stock_material_S1 = 0  # Stock de materiales del
proveedor S1
            self.stock_material_S2 = 0  # Stock de materiales del
proveedor S2
            self.stock_material_S3 = 0  # Stock de materialer del
proveedor S3

            self.cantidad_de_orden_de_produccion_PRODU = 0
            self.tiempo_de_produccion = []
            self.ordenes_de_produccion = []  # Inicializa una lista para
almacenar las órdenes de producción
            self.nuevas_ordenes_de_produccion = []
            self.numero_orden = 1  # Inicializa el número de orden en 1

            self.ordenes_pendientes_MAYO = []  # Lista para almacenar la
órden de compra que le generó el mayorista
            self.nuevas_ordenes_pendientes_MAYO = []

            self.dia_sin_stock_produ = False   # Inicializar la bandera
dia_sin_stock_produ
            self.historial_dias_sin_stock_produ = []
            self.contador_dias_sin_stock = 0

            self.punto_de_reorden = 0
            self.cantidad_disponible_PRODU = 0
            self.stock_material_PENDIENTE_a_recibir_de_S1 = 0  # Stock de
materia prima PENDIENTE del proveedor S1
            self.stock_material_PENDIENTE_a_recibir_de_S2 = 0  # Stock de
materia prima PENDIENTE del proveedor S2
            self.stock_material_PENDIENTE_a_recibir_de_S3 = 0  # Stock de
materia prima PENDIENTE del proveedor S3
            self.dias_para_recibir = 0
```

```
            self.total_pedidos_pendientes_MAYO = 0

        def calculo_tiempo_entrega(self):
            return random.randint(self.min_tiempo_entrega,
self.max_tiempo_entrega)

        def calculo_tiempo_de_produccion(self):
            return random.randint(self.min_tiempo_pcc,
self.max_tiempo_pcc)

        def produccion_PRODU(self):
            print(f"{negrita}stock producto TERMINADO productor INICIO
DIA:{fin_negrita}",  self.stock_producto_TERMINADO_productor)
            print("stock material proveedor S1 que tiene el productor
INICIO DIA:", self.stock_material_S1)
            print("stock material proveedor S2 que tiene el productor
INICIO DIA:", self.stock_material_S2)
            print("stock material proveedor S3 que tiene el productor
INICIO DIA:", self.stock_material_S3)

            # Calcular la cantidad de material disponible
            material_disponible = min(self.stock_material_S1,
self.stock_material_S2, self.stock_material_S3)
            print("material disponible:", material_disponible)
            print("stock producto en PROCESO productor INICIO DIA:",
self.stock_producto_en_proceso_PRODU)

            if material_disponible >= 1 and
self.stock_producto_en_proceso_PRODU < self.max_limite_pcc:

                # Calcular la cantidad de material adicional que se puede
procesar (capacidad_para_nuevas_ordenes)
                self.cantidad_de_orden_de_produccion_PRODU =
min(material_disponible, self.max_limite_pcc -
self.stock_producto_en_proceso_PRODU)
                print("cantidad adicional a PRODUCIR:",
self.cantidad_de_orden_de_produccion_PRODU)

                # Restar la cantidad de la orden de producción del
material disponible
                self.stock_material_S1 -=
self.cantidad_de_orden_de_produccion_PRODU
                self.stock_material_S2 -=
self.cantidad_de_orden_de_produccion_PRODU
```

```
                self.stock_material_S3 -=
self.cantidad_de_orden_de_produccion_PRODU
                print("stock material proveedor S1 que tiene el productor
SIN PROCESAR CON HOY:", self.stock_material_S1)
                print("stock material proveedor S2 que tiene el productor
SIN PROCESAR CON HOY:", self.stock_material_S2)
                print("stock material proveedor S3 que tiene el productor
SIN PROCESAR CON HOY:", self.stock_material_S3)

                # Sumar la cantidad de la orden de producción al material
en proceso
                self.stock_producto_en_proceso_PRODU +=
self.cantidad_de_orden_de_produccion_PRODU
                print("stock producto en PROCESO DESPUES DE PRODUCCION:",
self.stock_producto_en_proceso_PRODU)

                # Calcular el tiempo de producción
                self.tiempo_de_produccion =
self.calculo_tiempo_de_produccion()  # Obtener el tiempo de producción
                print("tiempo de produccion produ:",
self.tiempo_de_produccion)

                # Almacenar la orden de producción en la lista con el
número de orden, tiempo de producción y cantidad de la orden de producción
                self.ordenes_de_produccion.append({
                    'numero_orden': self.numero_orden,
                    'tiempo_de_produccion': self.tiempo_de_produccion,
                    'cantidad_orden':
self.cantidad_de_orden_de_produccion_PRODU
                })
                self.numero_orden += 1  # Incrementa el número de orden

            # Iterar sobre las órdenes de producción existentes y
actualizar el tiempo restante de producción
            nuevas_ordenes_de_produccion = []
            for orden in self.ordenes_de_produccion:
                tiempo_restante = orden['tiempo_de_produccion']
                if tiempo_restante > 1:
                    tiempo_restante -= 1
                    orden['tiempo_de_produccion'] = tiempo_restante
                    nuevas_ordenes_de_produccion.append(orden)
                else:
                    self.stock_producto_en_proceso_PRODU -=
orden['cantidad_orden']
```

```
                    self.stock_producto_TERMINADO_productor +=
orden['cantidad_orden']
            print(f"{negrita}stock_producto en PROCESO FIN DE
DIA:{fin_negrita}", self.stock_producto_en_proceso_PRODU)
            print(f"{negrita}stock producto TERMINADO FIN DE
DIA:{fin_negrita}",  self.stock_producto_TERMINADO_productor)
            self.ordenes_de_produccion = nuevas_ordenes_de_produccion

        # La orden de pedido que le coloca el mayorista al productor

        def productor_recibe_orden_de_compra_del_mayorista(self,
nombre_mayorista, orden_mayorista):

            nombre_mayorista_recibido_por_productor =
orden_mayorista['mayorista']
            cantidad_ordenada_mayo_recibido_por_produ =
orden_mayorista['cantidad']
            dias_tiempo_entrega_produ =
orden_mayorista['dias_tiempo_entrega']
            nombre_productor_recibido_por_produ =
orden_mayorista['productor']

            # Almacena la orden en la lista de órdenes pendientes, en
estado NO_PROCESADA
            self.ordenes_pendientes_MAYO.append({
                'mayorista': nombre_mayorista_recibido_por_productor,
                'cantidad': cantidad_ordenada_mayo_recibido_por_produ,
                'dias_tiempo_entrega': dias_tiempo_entrega_produ,
                'productor': nombre_productor_recibido_por_produ,
                'estado': 'NO_PROCESADA',
                'dias_espera': 0
            })

        def productor_procesa_ordenes_pendientes_del_mayorista(self):
            self.dia_sin_stock_produ = False  # Inicializar la bandera
            self.nuevas_ordenes_pendientes_MAYO = []

            for i, orden in enumerate(self.ordenes_pendientes_MAYO):
                orden['dias_espera'] += 1  # Incrementa los días de espera
en 1
                if orden['estado'] == 'NO_PROCESADA':
                    if orden['cantidad'] <=
self.stock_producto_TERMINADO_productor:
                        self.stock_producto_TERMINADO_productor -=
orden['cantidad']
```

```
                        orden['estado'] = 'PROCESADA'  # Cambio
NO_PROCESADA por PROCESADA
                    elif self.stock_producto_TERMINADO_productor > 0:
                        # Despachar parcialmente
                        cantidad_despachada =
self.stock_producto_TERMINADO_productor
                        orden_restante = {
                          'mayorista': orden['mayorista'],
                          'cantidad':  orden['cantidad'] -
cantidad_despachada,
                          'dias_tiempo_entrega':
orden['dias_tiempo_entrega'],
                          'productor': orden['productor'],
                          'estado': 'NO_PROCESADA',
                          'dias_espera': orden['dias_espera']
                        }
                        # Actualizar la orden actual con la cantidad
despachada
                        orden['cantidad'] = cantidad_despachada
                        orden['estado'] = 'PROCESADA'
                        self.stock_producto_TERMINADO_productor = 0
                        # Insertar la orden restante después de la actual
                        self.ordenes_pendientes_MAYO.insert(i + 1,
orden_restante)

                        # Actualizar la bandera
                        self.dia_sin_stock_produ = True
                    else:
                        # No hay suficiente stock para procesar la orden
                        self.dia_sin_stock_produ = True
                        continue

            self.historial_dias_sin_stock_produ.append(self.dia_sin_stock_
produ)
            #print(f"historial_dias_sin_stock de {self.nombre}:",
self.historial_dias_sin_stock_produ)

            # Incrementar el contador de días sin stock si hubo al menos
una venta perdida en el día
            if self.dia_sin_stock_produ:
                self.contador_dias_sin_stock += 1

            # Mostrar el contador actualizado
            print(f"contador_dias_sin_stock de {self.nombre}:",
self.contador_dias_sin_stock)
```

```
            for orden in self.ordenes_pendientes_MAYO:
                if orden['dias_tiempo_entrega'] == 0:
                    if orden['estado'] == 'PROCESADA':
                        cantidad_producto_enviadoMayoF = orden['cantidad']
                        nombre_mayorista = orden['mayorista']  # Obtener
el nombre del mayorista directamente del pedido
                        mayorista = self.next_agentes[nombre_mayorista]
                        mayorista.recibe_producto_del_productor(self.nombr
e, cantidad_producto_enviadoMayoF)
                    elif orden['estado'] == 'NO_PROCESADA':
                          self.nuevas_ordenes_pendientes_MAYO.append(orden
)  # Agregar el pedido a la nueva lista de pedidos pendientes
                elif orden['dias_tiempo_entrega'] > 0:
                    if orden['estado'] == 'PROCESADA':
                        orden['dias_tiempo_entrega'] -= 1  # Restar 1 a
los días de entrega
                        self.nuevas_ordenes_pendientes_MAYO.append(orden)
                    if orden['estado'] == 'NO_PROCESADA':
                        self.nuevas_ordenes_pendientes_MAYO.append(orden)

            # Reemplazar la lista original de pedidos pendientes con la
nueva lista
            self.ordenes_pendientes_MAYO =
self.nuevas_ordenes_pendientes_MAYO

            self.total_pedidos_pendientes_MAYO = sum(orden['cantidad'] for
orden in self.ordenes_pendientes_MAYO if orden['estado'] ==
'NO_PROCESADA')
            print(f"Total pedidos pendientes NO PROCESADOS del mayorista:
{self.total_pedidos_pendientes_MAYO}")

        #Lo que necesita el productor del proveedor

        def punto_de_reorden_PRODU(self):
            # Calcular los tiempos de entrega medios de los tres
materiales (distribución uniforme)
            tiempo_entrega_promedio_S1 = (S1.min_tiempo_entrega +
S1.max_tiempo_entrega) / 2
            print("Tiempo de entrega promedio provee_S1:",
tiempo_entrega_promedio_S1)
            tiempo_entrega_promedio_S2 = (S2.min_tiempo_entrega +
S2.max_tiempo_entrega) / 2
            print("Tiempo de entrega promedio provee_S2:",
tiempo_entrega_promedio_S2)
```

```
            tiempo_entrega_promedio_S3 = (S3.min_tiempo_entrega +
S3.max_tiempo_entrega) / 2
            print("Tiempo de entrega promedio provee_S3:",
tiempo_entrega_promedio_S3)
            # Calcular el tiempo total de entrega considerando el cuello
de botella (el mayor tiempo de entrega)
            tiempo_de_entrega_promedio = max(tiempo_entrega_promedio_S1,
tiempo_entrega_promedio_S2, tiempo_entrega_promedio_S3)
            print("Tiempo de entrega promedio:",
tiempo_de_entrega_promedio)

            # Calcular la desviación estándar (σ_LT) del tiempo de entrega
(distribución uniforme)
            sigma_LT_S1 = (S1.max_tiempo_entrega - S1.min_tiempo_entrega)
/ (12 ** 0.5)
            sigma_LT_S2 = (S2.max_tiempo_entrega - S2.min_tiempo_entrega)
/ (12 ** 0.5)
            sigma_LT_S3 = (S3.max_tiempo_entrega - S3.min_tiempo_entrega)
/ (12 ** 0.5)
            # Desviación estándar total del tiempo de entrega (cuello de
botella)
            sigma_LT = max(sigma_LT_S1, sigma_LT_S2, sigma_LT_S3)
            print("Desviación estándar del tiempo de entrega:", sigma_LT)

            # Calcular el tiempo medio de producción (distribución
uniforme)
            tiempo_procesamiento_material_promedio = (self.min_tiempo_pcc
+ self.max_tiempo_pcc) / 2
            print("Tiempo de procesamiento de material:",
tiempo_procesamiento_material_promedio)

            # Calcular la desviación estándar del tiempo de producción
(distribución uniforme)
            sigma_procesamiento = (self.max_tiempo_pcc -
self.min_tiempo_pcc) / (12 ** 0.5)
            print("sigma_procesamiento:", sigma_procesamiento)

            tiempo_total = tiempo_de_entrega_promedio +
tiempo_procesamiento_material_promedio
            print("tiempo_total:", tiempo_total)

            # Calcular la desviación estándar total combinada (demanda,
tiempo de entrega y tiempo de procesamiento)
```

```
            sigma_total = ((tiempo_total * self.desv_demanda ** 2) +
(self.media_demanda ** 2 * (sigma_LT ** 2 + sigma_procesamiento ** 2))) **
0.5
            print("Desviación estándar total:", sigma_total)
            Z = 1.96  # Para un nivel de confianza del 95%

            # Calcular el punto de reorden (ROP)
            self.punto_de_reorden = int((self.media_demanda *
tiempo_total) + (Z * sigma_total))
            print("punto de reorden produ:", self.punto_de_reorden)

            print("stock producto TERMINADO productor:",
self.stock_producto_TERMINADO_productor)
            print("stock producto en PROCESO productor:",
self.stock_producto_en_proceso_PRODU)
            print("Total pedidos pendientes NO PROCESADOS del mayorista:",
self.total_pedidos_pendientes_MAYO)

        def calculo_cantidad_a_ordenar_al_proveedorS1(self):

            print("stock MATERIAL S1 Productor:", self.stock_material_S1)
            print("stock material PENDIENTE a recibir de S1:",
self.stock_material_PENDIENTE_a_recibir_de_S1)

            self.posicion_inventario =
self.stock_producto_TERMINADO_productor +
self.stock_producto_en_proceso_PRODU + self.stock_material_S1 +
self.stock_material_PENDIENTE_a_recibir_de_S1 -
self.total_pedidos_pendientes_MAYO
            print(f"{negrita}posicion_inventario S1:{fin_negrita}",
self.posicion_inventario)
            if  self.posicion_inventario <= self.punto_de_reorden:
                self.cantidad_ordenada_produ = self.nivel_objetivo -
self.posicion_inventario
            else:
                self.cantidad_ordenada_produ = 0
            return self.cantidad_ordenada_produ

        def calculo_cantidad_a_ordenar_al_proveedorS2(self):

            print("stock MATERIAL S2 Productor:", self.stock_material_S2)
            print("stock material PENDIENTE a recibir de S2:",
self.stock_material_PENDIENTE_a_recibir_de_S2)
```

```
            self.posicion_inventario =
self.stock_producto_TERMINADO_productor +
self.stock_producto_en_proceso_PRODU + self.stock_material_S2 +
self.stock_material_PENDIENTE_a_recibir_de_S2 -
self.total_pedidos_pendientes_MAYO
            print(f"{negrita}posicion_inventario S2:{fin_negrita}",
self.posicion_inventario)
            if  self.posicion_inventario <= self.punto_de_reorden:
                self.cantidad_ordenada_produ = self.nivel_objetivo -
self.posicion_inventario
            else:
                self.cantidad_ordenada_produ = 0
            return self.cantidad_ordenada_produ

        def calculo_cantidad_a_ordenar_al_proveedorS3(self):

            print("stock MATERIAL S3 Productor:", self.stock_material_S3)
            print("stock material PENDIENTE a recibir de S3:",
self.stock_material_PENDIENTE_a_recibir_de_S3)

            self.posicion_inventario =
self.stock_producto_TERMINADO_productor +
self.stock_producto_en_proceso_PRODU + self.stock_material_S3 +
self.stock_material_PENDIENTE_a_recibir_de_S3 -
self.total_pedidos_pendientes_MAYO
            print(f"{negrita}posicion_inventario S3:{fin_negrita}",
self.posicion_inventario)
            if  self.posicion_inventario <= self.punto_de_reorden:
                self.cantidad_ordenada_produ = self.nivel_objetivo -
self.posicion_inventario
            else:
                self.cantidad_ordenada_produ = 0
            return self.cantidad_ordenada_produ

        def generar_orden_al_proveedor(self, cantidad_a_ordenar_produc_s1,
cantidad_a_ordenar_produc_s2, cantidad_a_ordenar_produc_s3):
            productor_orden_s1 = {}
            productor_orden_s2 = {}
            productor_orden_s3 = {}

            if cantidad_a_ordenar_produc_s1 > 0:
                self.dias_para_recibir = S1.calculo_tiempo_entrega() #
Establecer el contador de días para recibir la mercancía
                productor_orden_s1 = {
                    'productor': self.nombre,
```

```
                    'cantidad': cantidad_a_ordenar_produc_s1,
                    'dias_tiempo_entrega': self.dias_para_recibir,
                    'proveedor': 'S1',
                }
                print("orden pendiente proveedor:", S1.nombre,
productor_orden_s1)
                S1.proveedor_recibe_orden_de_compra_del_productor(self.nom
bre, productor_orden_s1)

                # Actualiza la cantidad pendiente del proveedor
                self.stock_material_PENDIENTE_a_recibir_de_S1 +=
cantidad_a_ordenar_produc_s1 # Stock de material PENDIENTE del proveedor
S1
                print(f"{negrita}MATERIAL TOTAL que tiene PENDIENTE
{self.nombre} del proveedor S1:{fin_negrita}",
self.stock_material_PENDIENTE_a_recibir_de_S1)

            if cantidad_a_ordenar_produc_s2 > 0:
              self.dias_para_recibir = S2.calculo_tiempo_entrega() #
Establecer el contador de días para recibir la mercancía
              productor_orden_s2 = {
                    'productor': self.nombre,
                    'cantidad': cantidad_a_ordenar_produc_s2,
                    'dias_tiempo_entrega': self.dias_para_recibir,
                    'proveedor': 'S2',
              }
              print("orden pendiente proveedor:", S2.nombre,
productor_orden_s2)
              S2.proveedor_recibe_orden_de_compra_del_productor(self.nombr
e, productor_orden_s2)

              # Actualiza la cantidad que tiene pendiente por enviar el
proveedor
              self.stock_material_PENDIENTE_a_recibir_de_S2 +=
cantidad_a_ordenar_produc_s2 # Stock de material PENDIENTE del proveedor
S2
              print(f"{negrita}MATERIAL TOTAL que tiene PENDIENTE
{self.nombre} del proveedor S2:{fin_negrita}",
self.stock_material_PENDIENTE_a_recibir_de_S2)

            if cantidad_a_ordenar_produc_s3 > 0:
              self.dias_para_recibir = S3.calculo_tiempo_entrega() #
Establecer el contador de días para recibir la mercancía
              productor_orden_s3 = {
                    'productor': self.nombre,
```

```
                    'cantidad': cantidad_a_ordenar_produc_s3,
                    'dias_tiempo_entrega': self.dias_para_recibir,
                    'proveedor': 'S3',
                }
                print("orden pendiente proveedor", S3.nombre,
productor_orden_s3)
                S3.proveedor_recibe_orden_de_compra_del_productor(self.nombr
e, productor_orden_s3)

                # Actualiza la cantidad que tiene pendiente por enviar el
proveedor
                self.stock_material_PENDIENTE_a_recibir_de_S3 +=
cantidad_a_ordenar_produc_s3 # Stock de material PENDIENTE del proveedor
S3
                print(f"{negrita}MATERIAL TOTAL que tiene PENDIENTE
{self.nombre} del proveedor
S3:{fin_negrita}",  self.stock_material_PENDIENTE_a_recibir_de_S3)

        def productor_recibe_materiales_del_proveedor(self,
nombre_proveedor, cantidad_producto_enviadoProduF):
            print("productor que recibe:", self.nombre)  # Accede al
nombre del productor desde self
            print("cantidad recibida:", cantidad_producto_enviadoProduF)
            print("material recibido del proveedor", nombre_proveedor)
            # Ingresar al stock el material recibido de cada proveedor
            if nombre_proveedor == 'S1':
                self.stock_material_S1 += cantidad_producto_enviadoProduF
            elif nombre_proveedor == 'S2':
                self.stock_material_S2 += cantidad_producto_enviadoProduF
            elif nombre_proveedor == 'S3':
                self.stock_material_S3 += cantidad_producto_enviadoProduF
            print(f"{negrita}stock MATERIAL productor DESPUES DE RECIBIR
la orden:{fin_negrita}", self.stock_material_S1, self.stock_material_S2,
self.stock_material_S3 )
            # Actualizar la cantidad pendiente de cada proveedor
            if nombre_proveedor == 'S1':
                self.stock_material_PENDIENTE_a_recibir_de_S1 -=
cantidad_producto_enviadoProduF
            elif nombre_proveedor == 'S2':
                self.stock_material_PENDIENTE_a_recibir_de_S2 -=
cantidad_producto_enviadoProduF
            elif nombre_proveedor == 'S3':
                self.stock_material_PENDIENTE_a_recibir_de_S3 -=
cantidad_producto_enviadoProduF
```

```
    class mayorista:
        def __init__(self, nombre, media_demanda, desv_demanda,
min_tiempo_entrega, max_tiempo_entrega, min_tiempo_pcc, max_tiempo_pcc,
nivel_objetivo, stock_producto_en_proceso_MAYO,
stock_producto_TERMINADO_mayorista, max_limite_pcc):
            self.nombre = nombre
            self.media_demanda = media_demanda
            self.desv_demanda = desv_demanda
            self.min_tiempo_entrega = min_tiempo_entrega
            self.max_tiempo_entrega = max_tiempo_entrega
            self.min_tiempo_pcc = min_tiempo_pcc
            self.max_tiempo_pcc = max_tiempo_pcc
            self.nivel_objetivo = nivel_objetivo

            self.stock_producto_en_proceso_MAYO =
stock_producto_en_proceso_MAYO
            self.stock_producto_TERMINADO_mayorista =
stock_producto_TERMINADO_mayorista
            self.max_limite_pcc = max_limite_pcc

            self.ordenes_pendientes_MINO = []  # Lista para almacenar las
órdenes de compra pendientes que ha generado el minorista
            self.nuevas_ordenes_pendientes_MINO = []

            self.dia_sin_stock_mayo = False   # Inicialización la bandera
dia_sin_stock_mayo
            self.historial_dias_sin_stock_mayo = []
            self.contador_dias_sin_stock = 0

            self.punto_de_reorden = 0
            self.cantidad_PENDIENTE_a_recibir_de_los_productores = 0
            self.dias_para_recibir = 0

            self.total_pedidos_pendientes_MINO = 0
            self.posicion_inventario = 0

        def __str__(self):
            return f"mayorista: {self.nombre}"

        def calculo_tiempo_entrega(self):
            return random.randint(self.min_tiempo_entrega,
self.max_tiempo_entrega)
```

```
        def recibe_orden_del_minorista(self, orden_minorista):     # El
minorista ejecuta esta linea con esta llamada
mayorista.recibe_orden_del_minorista(self.nombre, orden_minorista)

            nombre_minorista_recibido_por_mayo =
orden_minorista['minorista']   # Toma la clave 'minorista' del diccionario
orden_minorista y lo asigna a una variable llamada
nombre_minorista_recibido_por_mayo o lo que es lo mismo,
nombre_minorista_recibido_por_mayo es una variable a la cual le estoy
asignando la clave minorista
            cantidad_ordenada_mino_recibido_por_mayo =
orden_minorista['cantidad'] #Toma la clave cantidad y lo asigna a una
variable llamada cantidad_ordenada_mino_recibido_por_mayo
            dias_tiempo_entrega_mayo =
orden_minorista['dias_tiempo_entrega']  # Toma la clave
dias_tiempo_entrega y lo asigna a una variable llamada
dias_tiempo_entrega_mayo
            nombre_mayorista_recibido_por_mayo =
orden_minorista['mayorista']  # Toma la clave mayorista y lo asigna a una
variable llamada nombre_mayorista_recibido por_mayo

            # Almacena la orden en la lista de órdenes pendientes, en
estado NO_PROCESADA
            self.ordenes_pendientes_MINO.append({
                'minorista': nombre_minorista_recibido_por_mayo,
                'cantidad': cantidad_ordenada_mino_recibido_por_mayo,
                'dias_tiempo_entrega': dias_tiempo_entrega_mayo,
                'mayorista': nombre_mayorista_recibido_por_mayo,
                'estado': 'NO_PROCESADA',
                'dias_espera': 0
            })

        def mayorista_procesa_ordenes_pendientes_del_minorista(self):
            self.dia_sin_stock_mayo = False # Inicializar la bandera
            self.nuevas_ordenes_pendientes_MINO = []

            for i, orden in enumerate(self.ordenes_pendientes_MINO):
                orden['dias_espera'] += 1  # Incrementa los días de espera
en 1
                if orden['estado'] == 'NO_PROCESADA':
                    if orden['cantidad'] <=
self.stock_producto_TERMINADO_mayorista:
                        # Procesar la orden completamente
                        self.stock_producto_TERMINADO_mayorista -=
orden['cantidad']
```

```
                        orden['estado'] = 'PROCESADA'  # Cambio
NO_PROCESADA por PROCESADA
                    elif self.stock_producto_TERMINADO_mayorista > 0:
                        # Despachar parcialmente
                        cantidad_despachada =
self.stock_producto_TERMINADO_mayorista
                        orden_restante = {
                            'minorista': orden['minorista'],
                            'cantidad': orden['cantidad'] -
cantidad_despachada,
                            'dias_tiempo_entrega':
orden['dias_tiempo_entrega'],
                            'mayorista': orden['mayorista'],
                            'estado': 'NO_PROCESADA',
                            'dias_espera': orden['dias_espera']
                        }
                        # Actualizar la orden actual con la cantidad
despachada
                        orden['cantidad'] = cantidad_despachada
                        orden['estado'] = 'PROCESADA'
                        self.stock_producto_TERMINADO_mayorista -=
cantidad_despachada
                        # Insertar la orden restante después de la actual
                        self.ordenes_pendientes_MINO.insert(i + 1,
orden_restante)

                        # Actualizar la bandera
                        self.dia_sin_stock_mayo = True
                    else:
                        # No hay suficiente stock para procesar la orden
                        self.dia_sin_stock_mayo = True
                        continue

            self.historial_dias_sin_stock_mayo.append(self.dia_sin_stock_m
ayo)
            #print(f"historial_dias_sin_stock de {self.nombre}:",
self.historial_dias_sin_stock_mayo)

            # Incrementar el contador de días sin stock si hubo al menos
una venta perdida en el día
            if self.dia_sin_stock_mayo:
                self.contador_dias_sin_stock += 1

            # Mostrar el contador actualizado
```

```
            print(f"contador_dias_sin_stock de {self.nombre}:",
self.contador_dias_sin_stock)

            for orden in self.ordenes_pendientes_MINO:
                if orden['dias_tiempo_entrega'] == 0:
                    if orden['estado'] == 'PROCESADA':
                        cantidad_producto_enviadoMinoF = orden['cantidad']
                        nombre_minorista = orden['minorista'] # Obtener el
nombre del minorista directamente del pedido
                        minorista = self.next_agentes[nombre_minorista]
                        minorista.recibe_producto_del_mayorista(self.nombr
e, cantidad_producto_enviadoMinoF)
                    elif orden['estado'] == 'NO_PROCESADA':
                          self.nuevas_ordenes_pendientes_MINO.append(orden
)
                elif orden['dias_tiempo_entrega'] > 0:
                    if orden['estado'] == 'PROCESADA':
                        orden['dias_tiempo_entrega'] -= 1
                        self.nuevas_ordenes_pendientes_MINO.append(orden)
                    if orden['estado'] == 'NO_PROCESADA':
                        self.nuevas_ordenes_pendientes_MINO.append(orden)

            # Reemplazar la lista original de pedidos pendientes con la
nueva lista
            self.ordenes_pendientes_MINO =
self.nuevas_ordenes_pendientes_MINO

            self.total_pedidos_pendientes_MINO = sum(orden['cantidad'] for
orden in self.ordenes_pendientes_MINO if orden['estado'] ==
'NO_PROCESADA')
            print(f"Total pedidos pendientes NO PROCESADOS del minorista:
{self.total_pedidos_pendientes_MINO}")

        #Lo que necesita el mayorista del productor

        def calculo_cantidad_a_ordenar_al_productor(self):
            T_promedio = (P1.min_tiempo_entrega + P1.max_tiempo_entrega) /
2  # Promedio del tiempo de entrega - Distribución uniforme
            print("Promedio del tiempo de entrega mayo:", T_promedio)
            sigma_LT = (P1.max_tiempo_entrega - P1.min_tiempo_entrega) /
(12 ** 0.5)  # Desviación estándar del tiempo de entrega uniforme
            print("Desviación estándar del tiempo de entrega:", sigma_LT)

            # ROP ajustado para incluir incertidumbre tanto en la demanda
como en el tiempo de entrega
```

```
            sigma_total = ((T_promedio * self.desv_demanda ** 2) +
(self.media_demanda ** 2 * sigma_LT ** 2)) ** 0.5
            print("Desviación estándar total:", sigma_total)
            Z = 1.96  # Valor Z para el nivel de confianza
            self.punto_de_reorden = int((self.media_demanda * T_promedio)
+ (Z * sigma_total))
            print("punto de reorden mayo:", self.punto_de_reorden)

            print("stock producto TERMINADO mayorista:",
self.stock_producto_TERMINADO_mayorista)
            print("cantidad PENDIENTE a recibir de los productores:",
self.cantidad_PENDIENTE_a_recibir_de_los_productores)
            print("Total pedidos pendientes NO PROCESADOS del minorista:",
self.total_pedidos_pendientes_MINO)

            self.posicion_inventario =
self.stock_producto_TERMINADO_mayorista +
self.cantidad_PENDIENTE_a_recibir_de_los_productores -
self.total_pedidos_pendientes_MINO
            print(f"{negrita}posicion_inventario:{fin_negrita}",
self.posicion_inventario)
            if self.posicion_inventario <= self.punto_de_reorden:
                self.cantidad_ordenada_mayo = self.nivel_objetivo -
self.posicion_inventario
            else:
                self.cantidad_ordenada_mayo = 0

            return self.cantidad_ordenada_mayo

        def generar_orden_al_productor(self, cantidad_a_ordenar_mayo_p1,
cantidad_a_ordenar_mayo_p2):
            mayorista_orden_p1 = {}
            mayorista_orden_p2 = {}

            if cantidad_a_ordenar_mayo_p1 > 0:
                dias_para_recibir = P1.calculo_tiempo_entrega() #
Establecer el contador de días para recibir la mercancía
                mayorista_orden_p1 = {
                    'mayorista': self.nombre,
                    'cantidad': cantidad_a_ordenar_mayo_p1,
                    'dias_tiempo_entrega': dias_para_recibir,
                    'productor': 'P1',
                }
                print("orden pendiente productor:", P1.nombre,
mayorista_orden_p1)
```

```
                # Actualiza la cantidad pendiente de los productores
                self.cantidad_PENDIENTE_a_recibir_de_los_productores +=
cantidad_a_ordenar_mayo_p1
                print(f"cantidad PENDIENTE del mayorista {self.nombre} a
recibir de los productores:",
self.cantidad_PENDIENTE_a_recibir_de_los_productores)

                # Coloca la orden al agente anterior
                P1.productor_recibe_orden_de_compra_del_mayorista(self.nom
bre, mayorista_orden_p1)

            if cantidad_a_ordenar_mayo_p2 > 0:
                dias_para_recibir = P2.calculo_tiempo_entrega()
                mayorista_orden_p2 = {
                    'mayorista': self.nombre,
                    'cantidad': cantidad_a_ordenar_mayo_p2,
                    'dias_tiempo_entrega': dias_para_recibir,
                    'productor': 'P2',
                }
                print("orden pendiente productor:", P2.nombre,
mayorista_orden_p2)

                # Actualiza la cantidad pendiente de los productores
                self.cantidad_PENDIENTE_a_recibir_de_los_productores +=
cantidad_a_ordenar_mayo_p2
                print(f"cantidad PENDIENTE del mayorista {self.nombre} a
recibir de los productores:",
self.cantidad_PENDIENTE_a_recibir_de_los_productores)

                # Coloca la orden al agente anterior
                P2.productor_recibe_orden_de_compra_del_mayorista(self.nom
bre, mayorista_orden_p2)

        def recibe_producto_del_productor(self, nombre_productor,
cantidad_producto_enviadoMayoF):
            print("mayorista que recibe:", self.nombre)  # Accede al
nombre del mayorista desde self
            print("cantidad recibida:", cantidad_producto_enviadoMayoF)
            print("producto recibido del productor", nombre_productor)
            # Ingresar al stock el producto terminado recibido del
productor
            self.stock_producto_TERMINADO_mayorista +=
cantidad_producto_enviadoMayoF
```

```
            print(f"{negrita}stock de producto TERMINADO mayorista DESPUES
DE RECIBIR la orden:{fin_negrita}",
self.stock_producto_TERMINADO_mayorista)
            # Actualiza la cantidad pendiente de los productores
            self.cantidad_PENDIENTE_a_recibir_de_los_productores -=
cantidad_producto_enviadoMayoF
            print("cantidad PENDIENTE TOTAL a recibir de los
productores:", self.cantidad_PENDIENTE_a_recibir_de_los_productores)

    class minorista:
        def __init__(self, nombre, media_demanda, desv_demanda,
min_tiempo_entrega, max_tiempo_entrega, min_tiempo_pcc, max_tiempo_pcc,
nivel_objetivo, stock_producto_en_proceso_MINO,
stock_producto_TERMINADO_minorista, max_limite_pcc):
            self.nombre = nombre
            self.media_demanda = media_demanda
            self.desv_demanda = desv_demanda
            self.min_tiempo_entrega = min_tiempo_entrega
            self.max_tiempo_entrega = max_tiempo_entrega
            self.min_tiempo_pcc = min_tiempo_pcc
            self.max_tiempo_pcc = max_tiempo_pcc
            self.nivel_objetivo = nivel_objetivo

            self.stock_producto_en_proceso_MINO =
stock_producto_en_proceso_MINO
            self.stock_producto_TERMINADO_minorista =
stock_producto_TERMINADO_minorista
            self.max_limite_pcc = max_limite_pcc

            self.lista_venta_pendiente = []  # Almacena la venta pendiente
junto con el contador de días

            self.dia_sin_stock_mino = False   # Inicializa la bandera
dia_sin_stock_mino
            self.historial_dias_sin_stock_mino = []
            self.contador_dias_sin_stock = 0

            self.punto_de_reorden = 0
            self.cantidad_PENDIENTE_a_recibir_del_mayorista = 0
            self.dias_para_recibir = 0 # Establece en 0 el contador de
dias para recibir la mercancía

            self.total_ventas_pendientes_CLIENTE = 0

        def calculo_tiempo_entrega(self):
```

```
            return random.randint(self.min_tiempo_entrega,
self.max_tiempo_entrega)

        #Demanda

        def demanda_venta(self):
            demandaMino = max(0,
int(random.normalvariate(self.media_demanda, self.desv_demanda)))

            venta_efectiva = min(demandaMino,
self.stock_producto_TERMINADO_minorista) # Limita la venta según el stock
de producto terminado disponible. Cada cliente compra un producto si está
disponible.
            print("unidades vendidas:", venta_efectiva)
            self.stock_producto_TERMINADO_minorista -= venta_efectiva  #
Se reduce el stock de producto terminado en la cantidad vendida

            # Calcular ventas perdidas
            venta_pendiente = max(0, demandaMino - venta_efectiva)
            print("venta pendiente:", venta_pendiente)

            if venta_pendiente > 0:
                self.lista_venta_pendiente.append({"unidades_de_venta_pend
iente": venta_pendiente, "dias_espera": 0})

            return demandaMino

        def procesar_venta_pendiente(self):
            nueva_lista_venta_pendiente = []
            self.dia_sin_stock_mino = False

            for venta in self.lista_venta_pendiente:
                if self.stock_producto_TERMINADO_minorista >=
venta["unidades_de_venta_pendiente"]:
                    self.stock_producto_TERMINADO_minorista -=
venta["unidades_de_venta_pendiente"]
                    print(f"Se cumple una venta pendiente de
{venta['unidades_de_venta_pendiente']} unidades.")
                else:
                    venta["dias_espera"] += 1
                    self.dia_sin_stock_mino = True  # Activar la bandera
si la venta no se pudo satisfacer
                    if venta["dias_espera"] == 1:  # Se define que un
cliente espera un día
```

```
                        print(f"Venta pendiente incumplida (el cliente ya
no espera más): {venta['unidades_de_venta_pendiente']} unidades después de
{venta['dias_espera']} días.")
                    else:
                        pass  # No se hace nada

            self.lista_venta_pendiente = nueva_lista_venta_pendiente

            self.total_ventas_pendientes_CLIENTE =
sum(venta["unidades_de_venta_pendiente"] for venta in
self.lista_venta_pendiente)
            print(f"Total de unidades de venta pendientes:
{self.total_ventas_pendientes_CLIENTE}")

            self.historial_dias_sin_stock_mino.append(self.dia_sin_stock_m
ino)
            #print(f"historial_dias_sin_stock de {self.nombre}:",
self.historial_dias_sin_stock_mino)

            # Incrementar el contador de días sin stock si hubo venta
perdida en el día
            if self.dia_sin_stock_mino:
                self.contador_dias_sin_stock += 1

            # Mostrar los contadores actualizados
            print(f"contador_dias_sin_stock de {self.nombre}:",
self.contador_dias_sin_stock)

        #Lo que necesita el minorista del mayorista

        def calculo_cantidad_a_ordenar_al_mayorista(self):
            T_promedio = (W1.min_tiempo_entrega + W1.max_tiempo_entrega) /
2  # Promedio del tiempo de entrega - Distribución uniforme
            print("Promedio del tiempo de entrega mayo:", T_promedio)
            sigma_LT = (W1.max_tiempo_entrega - W1.min_tiempo_entrega) /
(12 ** 0.5)  # Desviación estándar del tiempo de entrega - Distribución
uniforme
            print("Desviación estándar del tiempo de entrega:", sigma_LT)

            # ROP ajustado para incluir incertidumbre tanto en demanda
como en tiempo de entrega
            sigma_total = ((T_promedio * self.desv_demanda ** 2) +
(self.media_demanda ** 2 * sigma_LT ** 2)) ** 0.5
            print("Desviación estándar total:", sigma_total)
            Z = 1.96  # Valor Z para el nivel de confianza
```

```
            self.punto_de_reorden = int((self.media_demanda * T_promedio)
+ (Z * sigma_total))
            print("punto de reorden mino:", self.punto_de_reorden)

            print("stock producto TERMINADO minorista:",
self.stock_producto_TERMINADO_minorista)
            print("cantidad PENDIENTE a recibir del mayorista:",
self.cantidad_PENDIENTE_a_recibir_del_mayorista)
            print("Total de unidades de venta pendientes:",
self.total_ventas_pendientes_CLIENTE)

            self.posicion_inventario =
self.stock_producto_TERMINADO_minorista +
self.cantidad_PENDIENTE_a_recibir_del_mayorista -
self.total_ventas_pendientes_CLIENTE
            print(f"{negrita}posicion_inventario:{fin_negrita}",
self.posicion_inventario)
            if self.posicion_inventario <= self.punto_de_reorden:
                self.cantidad_ordenada_mino = self.nivel_objetivo -
self.posicion_inventario
            else:
                self.cantidad_ordenada_mino = 0
            return self.cantidad_ordenada_mino

        def generar_orden_al_mayorista(self, cantidad_ordenada_mino,
nombre_mayorista):

            mayoristaR = self.previous_agentes[nombre_mayorista]
            print(mayoristaR)
            orden_minorista = {}

            if cantidad_ordenada_mino > 0:
                dias_para_recibir = mayoristaR.calculo_tiempo_entrega()
#La variable dias_para_recibir se calcula usando el método
calculo_tiempo_entrega() de la instancia mayorista
                # Almacenamiento de la información del pedido en el
diccionario orden_minorista
                orden_minorista = {
                    'minorista': self.nombre,                    # A la
clave 'minorista' se le asigna el valor self.nombre
                    'cantidad': cantidad_ordenada_mino,          # A la
clave 'cantidad' se le asigna el valor cantidad_ordenada_mino
                    'dias_tiempo_entrega': dias_para_recibir,    # A la
clave 'dias_tiempo_entrega' se le asigna el valor dias_para_recibir
```

```
                    'mayorista': nombre_mayorista                # A la
clave 'mayorista' se le asigna el valor nombre_mayorista
                }
                print(f"orden pendiente de {self.nombre} a recibir del
mayorista:", orden_minorista)

                # Actualiza la cantidad pendiente del mayorista
                self.cantidad_PENDIENTE_a_recibir_del_mayorista +=
cantidad_ordenada_mino
                print(f"cantidad PENDIENTE de {self.nombre} a recibir del
mayorista:", self.cantidad_PENDIENTE_a_recibir_del_mayorista)

            # Coloca la orden al actor anterior                    # Está
llamando al método recibe_orden_del_minorista de la instancia de la clase
"Mayorista", pasando como argumentos el nombre del minorista (obtenido a
través de self.nombre) y la orden que el minorista está enviando.
            mayoristaR.recibe_orden_del_minorista(orden_minorista) #
mayorista es una instancia de la clase mayorista y self.nombre hace
referencia al atributo nombre de la instancia actual de la clase minorista
                                                                    #
orden_minorista es un parámetro que contiene la información de la orden
que el minorista está enviando al mayorista.

        def recibe_producto_del_mayorista(self, nombre_mayorista,
cantidad_producto_enviadoMinoF):
            print("minorista que recibe:", self.nombre)  # Accede al
nombre del minorista desde self
            print("cantidad recibida:", cantidad_producto_enviadoMinoF)
            print("producto recibido del mayorista", nombre_mayorista)
            # Ingresar al stock el producto terminado recibido del
mayorista
            self.stock_producto_TERMINADO_minorista +=
cantidad_producto_enviadoMinoF
            print(f"{negrita}stock de producto TERMINADO minorista DESPUES
DE RECIBIR la orden:{fin_negrita}",
self.stock_producto_TERMINADO_minorista)
            # Actualiza la cantidad pendiente del mayorista
            self.cantidad_PENDIENTE_a_recibir_del_mayorista -=
cantidad_producto_enviadoMinoF
            print("cantidad PENDIENTE a recibir del mayorista:",
self.cantidad_PENDIENTE_a_recibir_del_mayorista)

    desviacion_minorista_1 = 35
    print("Desviación estándar minorista_1:", desviacion_minorista_1)
    desviacion_minorista_2 = 35
```

```
    print("Desviación estándar minorista_2:", desviacion_minorista_2)
    desviacion_minorista_3 = 35
    print("Desviación estándar minorista_3:", desviacion_minorista_3)
    desviacion_mayorista_1 =  desviacion_minorista_1
    print("Desviación estándar mayorista_1:", desviacion_mayorista_1)
    desviacion_mayorista_2 = math.sqrt((desviacion_minorista_2)**2 +
(desviacion_minorista_3)**2)
    print("Desviación estándar mayorista_2:", desviacion_mayorista_2)
    desviacion_productor_1 = math.sqrt((0.5 * desviacion_mayorista_1)**2 +
(0.5 * desviacion_mayorista_2)**2)
    print("Desviación estándar productor_1:", desviacion_productor_1)
    desviacion_productor_2 = math.sqrt((0.5 * desviacion_mayorista_1)**2 +
(0.5 * desviacion_mayorista_2)**2)
    print("Desviación estándar productor_2:", desviacion_productor_2)
    desviacion_proveedor_1 = math.sqrt((desviacion_productor_1)**2 +
(desviacion_productor_2)**2)
    print("Desviación estándar proveedor_1:", desviacion_proveedor_1)
    desviacion_proveedor_2 = math.sqrt((desviacion_productor_1)**2 +
(desviacion_productor_2)**2)
    print("Desviación estándar proveedor_2:", desviacion_proveedor_2)
    desviacion_proveedor_3 = math.sqrt((desviacion_productor_1)**2 +
(desviacion_productor_2)**2)
    print("Desviación estándar proveedor_3:", desviacion_proveedor_3)

    # Crear instancias de las clases
    S1 = proveedor("S1", 750, desviacion_proveedor_1,
min_tiempo_entrega["S1"], max_tiempo_entrega["S1"], min_tiempo_pcc["S1"],
max_tiempo_pcc["S1"], nivel_objetivo["S1"], 0, 11000, 20000);
    S2 = proveedor("S2", 750, desviacion_proveedor_2,
min_tiempo_entrega["S2"], max_tiempo_entrega["S2"], min_tiempo_pcc["S2"],
max_tiempo_pcc["S2"], nivel_objetivo["S2"], 0, 11000, 20000);
    S3 = proveedor("S3", 750, desviacion_proveedor_3,
min_tiempo_entrega["S3"], max_tiempo_entrega["S3"], min_tiempo_pcc["S3"],
max_tiempo_pcc["S3"], nivel_objetivo["S3"], 0, 12000, 20000);
    P1 = productor("P1", 375, desviacion_productor_1,
min_tiempo_entrega["P1"], max_tiempo_entrega["P1"], min_tiempo_pcc["P1"],
max_tiempo_pcc["P1"], nivel_objetivo["P1"], 0, 3000, 20000);
    P2 = productor("P2", 375, desviacion_productor_2,
min_tiempo_entrega["P2"], max_tiempo_entrega["P2"], min_tiempo_pcc["P2"],
max_tiempo_pcc["P2"], nivel_objetivo["P2"], 0, 4000, 20000);
    W1 = mayorista("W1", 250, desviacion_mayorista_1,
min_tiempo_entrega["W1"], max_tiempo_entrega["W1"], min_tiempo_pcc["W1"],
max_tiempo_pcc["W1"], nivel_objetivo["W1"], 0, 2000, 0);
```

```
    W2 = mayorista("W2", 500, desviacion_mayorista_2,
min_tiempo_entrega["W2"], max_tiempo_entrega["W2"], min_tiempo_pcc["W2"],
max_tiempo_pcc["W2"], nivel_objetivo["W2"], 0, 4000, 0);
    R1 = minorista("R1",  250, desviacion_minorista_1,
min_tiempo_entrega["R1"], max_tiempo_entrega["R1"], min_tiempo_pcc["R1"],
max_tiempo_pcc["R1"], nivel_objetivo["R1"], 0, 1500, 0);
    R2 = minorista("R2", 250, desviacion_minorista_2,
min_tiempo_entrega["R2"], max_tiempo_entrega["R2"], min_tiempo_pcc["R2"],
max_tiempo_pcc["R2"], nivel_objetivo["R2"], 0, 1500, 0);
    R3 = minorista("R3",  250, desviacion_minorista_3,
min_tiempo_entrega["R3"], max_tiempo_entrega["R3"], min_tiempo_pcc["R3"],
max_tiempo_pcc["R3"], nivel_objetivo["R3"], 0, 1500, 0);

    # Configuracion de relaciones
    R1.previous_agentes = {"W1": W1}
    R2.previous_agentes = {"W2": W2}
    R3.previous_agentes = {"W2": W2}

    W1.next_agentes = {"R1": R1}
    W2.next_agentes = {"R2": R2, "R3": R3}

    W1.previous_agentes = {"P1": P1, "P2": P2}
    W2.previous_agentes = {"P1": P1, "P2": P2}

    P1.next_agentes = {"W1": W1, "W2": W2}
    P2.next_agentes = {"W1": W1, "W2": W2}

    P1.previous_agentes = {"S1": S1, "S2": S2, "S3": S3}
    P2.previous_agentes = {"S1": S1, "S2": S2, "S3": S3}

    S1.next_agentes = {"P1": P1, "P2": P2}
    S2.next_agentes = {"P1": P1, "P2": P2}
    S3.next_agentes = {"P1": P1, "P2": P2}

    # Crear lista de proveedores
    lista_proveedores = [S1, S2, S3]

    # Asignar la lista de proveedores a cada instancia de proveedor
    S1.lista_proveedores = lista_proveedores
    S2.lista_proveedores = lista_proveedores
    S3.lista_proveedores = lista_proveedores


    # Simular el sistema durante un número determinado de días
```

```
    for step in range(num_dias):
        print("~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~Dia:", step)

        print("***********************************************************************MINORISTA**********************************************************************")
        print(f"{negrita}*********{fin_negrita}")
        R1.procesar_venta_pendiente()
        R2.procesar_venta_pendiente()
        R3.procesar_venta_pendiente()
        print(f"{negrita}*********{fin_negrita}")

        print(f"{negrita}************************************************{fin_negrita}")
        print(f"{negrita}Colocación de pedidos del Minorista al Mayorista{fin_negrita}")
        print(f"{negrita}************************************************{fin_negrita}")

        print("R1")
        print(f"{negrita}Valor stock_producto_TERMINADO_minorista:{fin_negrita}", R1.stock_producto_TERMINADO_minorista)
        R1demandaMino = R1.demanda_venta()
        print(f"{negrita}Valor stock_producto_TERMINADO_minorista DESPUES DE LA VENTA:{fin_negrita}", R1.stock_producto_TERMINADO_minorista)
        R1candidad_orden = R1.calculo_cantidad_a_ordenar_al_mayorista()
        print(f"{negrita}Orden a colocar del minorista R1 al mayorista W1:{fin_negrita}","R1", R1candidad_orden, "W1")
        if R1candidad_orden > 0:
            R1.generar_orden_al_mayorista(R1candidad_orden, "W1") # Coloca los pedidos utilizando las cantidades de pedido calculadas
        else:
            print("R1 no coloca una orden al mayorista W1 en este día")

        print("R2")
        print(f"{negrita}Valor stock_producto_TERMINADO_minorista:{fin_negrita}", R2.stock_producto_TERMINADO_minorista)
        R2demandaMino = R2.demanda_venta()
        print(f"{negrita}Valor stock_producto_TERMINADO_minorista DESPUES DE LA VENTA:{fin_negrita}", R2.stock_producto_TERMINADO_minorista)
        R2candidad_orden = R2.calculo_cantidad_a_ordenar_al_mayorista()
```

```
        print(f"{negrita}Orden a colocar del minorista R2 al mayorista
W2:{fin_negrita}", "R2", R2candidad_orden, "W2")
        if R2candidad_orden > 0:
            R2.generar_orden_al_mayorista(R2candidad_orden, "W2") # Coloca
los pedidos utilizando las cantidades de pedido calculadas
        else:
            print("R2 no coloca una orden al mayorista W2 en este día")

        print("R3")
        print(f"{negrita}Valor
stock_producto_TERMINADO_minorista:{fin_negrita}",
R3.stock_producto_TERMINADO_minorista)
        R3demandaMino= R3.demanda_venta()
        print(f"{negrita}Valor stock_producto_TERMINADO_minorista DESPUES
DE LA VENTA:{fin_negrita}", R3.stock_producto_TERMINADO_minorista)
        R3candidad_orden = R3.calculo_cantidad_a_ordenar_al_mayorista()
        print(f"{negrita}Orden a colocar del minorista R3 al mayorista
W2:{fin_negrita}", "R3", R3candidad_orden, "W2")
        if R3candidad_orden > 0:
            R3.generar_orden_al_mayorista(R3candidad_orden, "W2") # Coloca
los pedidos utilizando las cantidades de pedido calculadas
        else:
            print("R3 no coloca una orden al mayorista W2 en este día")

        print("**********************************************************
******MAYORISTA**************************************************
******")
        print(f"{negrita}*********{fin_negrita}")
        W1.mayorista_procesa_ordenes_pendientes_del_minorista()
        W2.mayorista_procesa_ordenes_pendientes_del_minorista()
        print(f"{negrita}*********{fin_negrita}")

        print(f"{negrita}**********************************************{
fin_negrita}")
        print(f"{negrita}Colocación de pedidos del Mayorista a los
Productores{fin_negrita}")
        print(f"{negrita}**********************************************{
fin_negrita}")

        print("W1")
        print(f"{negrita}Valor_stock_producto_TERMINADO_mayorista:{fin_neg
rita}", W1.stock_producto_TERMINADO_mayorista)
        W1cantidad_orden = W1.calculo_cantidad_a_ordenar_al_productor()
```

```
        print(f"{negrita}Resultado de
calculo_cantidad_a_ordenar_al_productor por parte de
W1:{fin_negrita}",  W1cantidad_orden)
        if W1cantidad_orden > 0:
            cantidad_a_ordenar_mayo_p1 = int(W1cantidad_orden * 0.5)
            cantidad_a_ordenar_mayo_p2 = int(W1cantidad_orden * 0.5)
            print(f"{negrita}Orden a colocar del mayorista W1 a los
productores:{fin_negrita}", cantidad_a_ordenar_mayo_p1, "P1",
cantidad_a_ordenar_mayo_p2, "P2")
            W1.generar_orden_al_productor(cantidad_a_ordenar_mayo_p1,
cantidad_a_ordenar_mayo_p2)
        else:
            print("W1 no coloca órdenes a los productores")

        print("W2")
        print(f"{negrita}Valor_stock_producto_TERMINADO_mayorista:{fin_neg
rita}", W2.stock_producto_TERMINADO_mayorista)
        W2cantidad_orden = W2.calculo_cantidad_a_ordenar_al_productor()
        print(f"{negrita}Resultado de
calculo_cantidad_a_ordenar_al_productor por parte de
W2:{fin_negrita}",  W2cantidad_orden)
        if W2cantidad_orden > 0:
            cantidad_a_ordenar_mayo_p1 = int(W2cantidad_orden * 0.5)
            cantidad_a_ordenar_mayo_p2 = int(W2cantidad_orden * 0.5)
            print(f"{negrita}Orden a colocar del mayorista W2 a los
productores:{fin_negrita}", cantidad_a_ordenar_mayo_p1, "P1",
cantidad_a_ordenar_mayo_p2, "P2")
            W2.generar_orden_al_productor(cantidad_a_ordenar_mayo_p1,
cantidad_a_ordenar_mayo_p2)
        else:
            print("W2 no coloca órdenes a los productores")

        print("**********************************************************
******PRODUCTOR**********************************************************
******")

        print(f"{negrita}**********************************************{
fin_negrita}")
        print(f"{negrita}Proceso de Producción del
Productor{fin_negrita}")
        print(f"{negrita}**********************************************{
fin_negrita}")

        print("P1")
        P1.produccion_PRODU()
```

```
        print("P2")
        P2.produccion_PRODU()

        print(f"{negrita}**********************************************{
fin_negrita}")
        print(f"{negrita}Colocación de pedidos del Productor a los
Proveedores{fin_negrita}")
        print(f"{negrita}**********************************************{
fin_negrita}")

        print(f"{negrita}*********{fin_negrita}")
        P1.productor_procesa_ordenes_pendientes_del_mayorista()
        P2.productor_procesa_ordenes_pendientes_del_mayorista()
        print(f"{negrita}*********{fin_negrita}")

        print("P1")
        P1punto_reorden_Produ = P1.punto_de_reorden_PRODU()
        P1cantidad_ordenS1 =
P1.calculo_cantidad_a_ordenar_al_proveedorS1()
        print(f"{negrita}Resultado de
calculo_cantidad_a_ordenar_al_proveedor S1 por parte de
P1:{fin_negrita}",  P1cantidad_ordenS1)
        P1cantidad_ordenS2 =
P1.calculo_cantidad_a_ordenar_al_proveedorS2()
        print(f"{negrita}Resultado de
calculo_cantidad_a_ordenar_al_proveedor S2 por parte de
P1:{fin_negrita}",  P1cantidad_ordenS2)
        P1cantidad_ordenS3 =
P1.calculo_cantidad_a_ordenar_al_proveedorS3()
        print(f"{negrita}Resultado de
calculo_cantidad_a_ordenar_al_proveedor S3 por parte de
P1:{fin_negrita}",  P1cantidad_ordenS3)

        P1.generar_orden_al_proveedor(P1cantidad_ordenS1,
P1cantidad_ordenS2, P1cantidad_ordenS3)

        print("P2")
        P2punto_reorden_Produ = P2.punto_de_reorden_PRODU()
        P2cantidad_ordenS1 =
P2.calculo_cantidad_a_ordenar_al_proveedorS1()
        print(f"{negrita}Resultado de
calculo_cantidad_a_ordenar_al_proveedor S1 por parte de
P2:{fin_negrita}",  P2cantidad_ordenS1)
        P2cantidad_ordenS2 =
P2.calculo_cantidad_a_ordenar_al_proveedorS2()
```

```
        print(f"{negrita}Resultado de
calculo_cantidad_a_ordenar_al_proveedor S2 por parte de
P2:{fin_negrita}",  P2cantidad_ordenS2)
        P2cantidad_ordenS3 =
P2.calculo_cantidad_a_ordenar_al_proveedorS3()
        print(f"{negrita}Resultado de
calculo_cantidad_a_ordenar_al_proveedor S3 por parte de
P2:{fin_negrita}",  P2cantidad_ordenS3)

        P2.generar_orden_al_proveedor(P2cantidad_ordenS1,
P2cantidad_ordenS2, P2cantidad_ordenS3)

        print("**********************************************************
******PROVEEDOR**********************************************************
******")

        print(f"{negrita}**********************************************{
fin_negrita}")
        print(f"{negrita}Proceso de producción del
Proveedor{fin_negrita}")
        print(f"{negrita}**********************************************{
fin_negrita}")

        print("S1")
        S1.produccion_PROVEE()
        print("S2")
        S2.produccion_PROVEE()
        print("S3")
        S3.produccion_PROVEE()

        print(f"{negrita}**********************************************{
fin_negrita}")
        print(f"{negrita}Necesidades del Proveedor{fin_negrita}")
        print(f"{negrita}**********************************************{
fin_negrita}")

        print(f"{negrita}*********{fin_negrita}")
        S1.proveedor_procesa_ordenes_pendientes_del_productor()
        S2.proveedor_procesa_ordenes_pendientes_del_productor()
        S3.proveedor_procesa_ordenes_pendientes_del_productor()
        print(f"{negrita}*********{fin_negrita}")

        print("S1")
        S1cantidad_orden = S1.calculo_cantidad_necesaria_mp()
```

```
        print(f"{negrita}Resultado de calculo_cantidad_necesaria por parte
de S1:{fin_negrita}",  S1cantidad_orden)
        if S1cantidad_orden > 0:
            S1.generar_orden_mp(S1cantidad_orden)
        else:
            print("S1 no necesita mp")

        print("S2")
        S2cantidad_orden = S2.calculo_cantidad_necesaria_mp()
        print(f"{negrita}Resultado de calculo_cantidad_necesaria por parte
de S2:{fin_negrita}",  S2cantidad_orden)
        if S2cantidad_orden > 0:
            S2.generar_orden_mp(S2cantidad_orden)
        else:
            print("S2 no necesita mp")

        print("S3")
        S3cantidad_orden = S3.calculo_cantidad_necesaria_mp()
        print(f"{negrita}Resultado de calculo_cantidad_necesaria por parte
de S3:{fin_negrita}",  S3cantidad_orden)
        if S3cantidad_orden > 0:
            S3.generar_orden_mp(S3cantidad_orden)
        else:
            print("S3 no necesita mp")

        S1.procesar_orden_mp()
        S2.procesar_orden_mp()
        S3.procesar_orden_mp()

    #*****************************************************************
** Para S1, S2, S3
    # Inicializa los contadores de eventos relevantes para el calculo de
probabilidades condicionales de cada réplica
    total_S1_op = 0
    total_S1_no_op = 0
    total_S2_op = 0
    total_S2_no_op = 0
    total_S3_op = 0
    total_S3_no_op = 0

    # Iteración a través de los días y contar eventos relevantes (Si el
elemento en la posición i de S1.historial_dias_sin_stock_provee es False.
    # not invierte el valor booleano. Si
S1.historial_dias_sin_stock_provee[i] es False, entonces not
S1.historial_dias_sin_stock_provee[i] será True.)
```

```
    for i in range(num_dias):
        # Determinar estado operativo o no operativo de S1 para el día i
de la réplica actual
        S1_op = 0 if not S1.historial_dias_sin_stock_provee[i] else 1
        # Determinar estado operativo o no operativo de S2 para el día i
de la réplica actual
        S2_op = 0 if not S2.historial_dias_sin_stock_provee[i] else 1
        # Determinar estado operativo o no operativo de S3 para el día i
de la réplica actual
        S3_op = 0 if not S3.historial_dias_sin_stock_provee[i] else 1

        # Contar eventos relevantes para S1
        if S1_op == 0:
            total_S1_op += 1
        else:
            total_S1_no_op += 1

        # Contar eventos relevantes para S2
        if S2_op == 0:
            total_S2_op += 1
        else:
            total_S2_no_op += 1

        # Contar eventos relevantes para S3
        if S3_op == 0:
            total_S3_op += 1
        else:
            total_S3_no_op += 1

    # Calcular probabilidades operativas y no operativas para esta réplica
    probabilidad_operativo_S1 = total_S1_op / num_dias
    probabilidad_no_operativo_S1 = total_S1_no_op / num_dias
    probabilidad_operativo_S2 = total_S2_op / num_dias
    probabilidad_no_operativo_S2 = total_S2_no_op / num_dias
    probabilidad_operativo_S3 = total_S3_op / num_dias
    probabilidad_no_operativo_S3 = total_S3_no_op / num_dias

    # Crear la matriz de probabilidades de S1 para la réplica
    matriz_probabilidades_S1 = np.array([[probabilidad_operativo_S1],
[probabilidad_no_operativo_S1]])
    # Agregar la matriz a la lista de resultados de las réplicas
    matrices_probabilidades_replicas_S1.append(matriz_probabilidades_S1)

    # Crear la matriz de probabilidades de S2 para la réplica
```

```
    matriz_probabilidades_S2 = np.array([[probabilidad_operativo_S2],
[probabilidad_no_operativo_S2]])
    # Agregar la matriz a la lista de resultados de las réplicas
    matrices_probabilidades_replicas_S2.append(matriz_probabilidades_S2)

    # Crear la matriz de probabilidades de S3 para la réplica
    matriz_probabilidades_S3 = np.array([[probabilidad_operativo_S3],
[probabilidad_no_operativo_S3]])
    # Agregar la matriz a la lista de resultados de las réplicas
    matrices_probabilidades_replicas_S3.append(matriz_probabilidades_S3)

    # Verificación de la suma de probabilidades. Debe ser igual a 1 ya que
total_SX_op + total_SX_no_op = num_dias.
    assert probabilidad_operativo_S1 + probabilidad_no_operativo_S1 == 1.0
    assert probabilidad_operativo_S2 + probabilidad_no_operativo_S2 == 1.0
    assert probabilidad_operativo_S3 + probabilidad_no_operativo_S3 == 1.0

    #*****************************************************************
** Para P1
    # Inicializa los contadores de eventos relevantes para el calculo de
probabilidades condicionales de cada réplica para P1
    total_S1_op_S2_op_S3_op = 0
    total_S1_op_S2_op_S3_no_op = 0
    total_S1_op_S2_no_op_S3_op = 0
    total_S1_op_S2_no_op_S3_no_op= 0
    total_S1_no_op_S2_op_S3_op = 0
    total_S1_no_op_S2_op_S3_no_op = 0
    total_S1_no_op_S2_no_op_S3_op = 0
    total_S1_no_op_S2_no_op_S3_no_op = 0

    total_S1_op_S2_op_S3_op_P1_op = 0
    total_S1_op_S2_op_S3_no_op_P1_op = 0
    total_S1_op_S2_no_op_S3_op_P1_op = 0
    total_S1_op_S2_no_op_S3_no_op_P1_op = 0
    total_S1_no_op_S2_op_S3_op_P1_op = 0
    total_S1_no_op_S2_op_S3_no_op_P1_op = 0
    total_S1_no_op_S2_no_op_S3_op_P1_op = 0
    total_S1_no_op_S2_no_op_S3_no_op_P1_op = 0

    # Iterar a través de los días y contar eventos relevantes
    for i in range(num_dias):
        S1_op = 0 if not S1.historial_dias_sin_stock_provee[i] else 1
        S2_op = 0 if not S2.historial_dias_sin_stock_provee[i] else 1
        S3_op = 0 if not S3.historial_dias_sin_stock_provee[i] else 1
        P1_op = 0 if not P1.historial_dias_sin_stock_produ[i] else 1
```

```
        # Contar combinaciones de estados de S1, S2, S3
        if S1_op == 0 and S2_op == 0 and S3_op == 0:
            total_S1_op_S2_op_S3_op += 1
            if P1_op == 0:
                total_S1_op_S2_op_S3_op_P1_op += 1
        elif S1_op == 0 and S2_op == 0 and S3_op == 1:
            total_S1_op_S2_op_S3_no_op += 1
            if P1_op == 0:
                total_S1_op_S2_op_S3_no_op_P1_op += 1
        elif S1_op == 0 and S2_op == 1 and S3_op == 0:
            total_S1_op_S2_no_op_S3_op += 1
            if P1_op == 0:
                total_S1_op_S2_no_op_S3_op_P1_op += 1
        elif S1_op == 0 and S2_op == 1 and S3_op == 1:
            total_S1_op_S2_no_op_S3_no_op += 1
            if P1_op == 0:
                total_S1_op_S2_no_op_S3_no_op_P1_op += 1
        elif S1_op == 1 and S2_op == 0 and S3_op == 0:
            total_S1_no_op_S2_op_S3_op += 1
            if P1_op == 0:
                total_S1_no_op_S2_op_S3_op_P1_op += 1
        elif S1_op == 1 and S2_op == 0 and S3_op == 1:
            total_S1_no_op_S2_op_S3_no_op += 1
            if P1_op == 0:
                total_S1_no_op_S2_op_S3_no_op_P1_op += 1
        elif S1_op == 1 and S2_op == 1 and S3_op == 0:
            total_S1_no_op_S2_no_op_S3_op += 1
            if P1_op == 0:
                total_S1_no_op_S2_no_op_S3_op_P1_op += 1
        elif S1_op == 1 and S2_op == 1 and S3_op == 1:
            total_S1_no_op_S2_no_op_S3_no_op += 1
            if P1_op == 0:
                total_S1_no_op_S2_no_op_S3_no_op_P1_op += 1

    # Calcular probabilidades condicionales
    prob_P1_op_dado_S1_op_S2_op_S3_op = total_S1_op_S2_op_S3_op_P1_op /
total_S1_op_S2_op_S3_op if total_S1_op_S2_op_S3_op > 0 else 0.0
    prob_P1_op_dado_S1_op_S2_op_S3_no_op =
total_S1_op_S2_op_S3_no_op_P1_op / total_S1_op_S2_op_S3_no_op if
total_S1_op_S2_op_S3_no_op > 0 else 0.0
    prob_P1_op_dado_S1_op_S2_no_op_S3_op =
total_S1_op_S2_no_op_S3_op_P1_op / total_S1_op_S2_no_op_S3_op if
total_S1_op_S2_no_op_S3_op > 0 else 0.0
```

```
    prob_P1_op_dado_S1_op_S2_no_op_S3_no_op =
total_S1_op_S2_no_op_S3_no_op_P1_op / total_S1_op_S2_no_op_S3_no_op if
total_S1_op_S2_no_op_S3_no_op > 0 else 0.0
    prob_P1_op_dado_S1_no_op_S2_op_S3_op =
total_S1_no_op_S2_op_S3_op_P1_op / total_S1_no_op_S2_op_S3_op if
total_S1_no_op_S2_op_S3_op > 0 else 0.0
    prob_P1_op_dado_S1_no_op_S2_op_S3_no_op =
total_S1_no_op_S2_op_S3_no_op_P1_op / total_S1_no_op_S2_op_S3_no_op if
total_S1_no_op_S2_op_S3_no_op > 0 else 0.0
    prob_P1_op_dado_S1_no_op_S2_no_op_S3_op =
total_S1_no_op_S2_no_op_S3_op_P1_op / total_S1_no_op_S2_no_op_S3_op if
total_S1_no_op_S2_no_op_S3_op > 0 else 0.0
    prob_P1_op_dado_S1_no_op_S2_no_op_S3_no_op =
total_S1_no_op_S2_no_op_S3_no_op_P1_op / total_S1_no_op_S2_no_op_S3_no_op
if total_S1_no_op_S2_no_op_S3_no_op > 0 else 0.0

    # Calcular probabilidades complementarias
    prob_P1_no_op_dado_S1_op_S2_op_S3_op = 1 -
prob_P1_op_dado_S1_op_S2_op_S3_op
    prob_P1_no_op_dado_S1_op_S2_op_S3_no_op = 1 -
prob_P1_op_dado_S1_op_S2_op_S3_no_op
    prob_P1_no_op_dado_S1_op_S2_no_op_S3_op  = 1 -
prob_P1_op_dado_S1_op_S2_no_op_S3_op
    prob_P1_no_op_dado_S1_op_S2_no_op_S3_no_op = 1 -
prob_P1_op_dado_S1_op_S2_no_op_S3_no_op
    prob_P1_no_op_dado_S1_no_op_S2_op_S3_op = 1 -
prob_P1_op_dado_S1_no_op_S2_op_S3_op
    prob_P1_no_op_dado_S1_no_op_S2_op_S3_no_op = 1 -
prob_P1_op_dado_S1_no_op_S2_op_S3_no_op
    prob_P1_no_op_dado_S1_no_op_S2_no_op_S3_op = 1 -
prob_P1_op_dado_S1_no_op_S2_no_op_S3_op
    prob_P1_no_op_dado_S1_no_op_S2_no_op_S3_no_op = 1 -
prob_P1_op_dado_S1_no_op_S2_no_op_S3_no_op

    # Crear la matriz de probabilidades de P1 para la réplica
    # Primera fila: probabilidades de que P1 esté operativo
    # Segunda fila: probabilidades de que P1 no esté operativo
    matriz_probabilidades_P1 = np.array([
        [
            prob_P1_op_dado_S1_op_S2_op_S3_op,
            prob_P1_op_dado_S1_op_S2_op_S3_no_op,
            prob_P1_op_dado_S1_op_S2_no_op_S3_op,
            prob_P1_op_dado_S1_op_S2_no_op_S3_no_op,
            prob_P1_op_dado_S1_no_op_S2_op_S3_op,
            prob_P1_op_dado_S1_no_op_S2_op_S3_no_op,
```

```
            prob_P1_op_dado_S1_no_op_S2_no_op_S3_op,
            prob_P1_op_dado_S1_no_op_S2_no_op_S3_no_op
        ],
        [
            prob_P1_no_op_dado_S1_op_S2_op_S3_op,
            prob_P1_no_op_dado_S1_op_S2_op_S3_no_op,
            prob_P1_no_op_dado_S1_op_S2_no_op_S3_op,
            prob_P1_no_op_dado_S1_op_S2_no_op_S3_no_op,
            prob_P1_no_op_dado_S1_no_op_S2_op_S3_op,
            prob_P1_no_op_dado_S1_no_op_S2_op_S3_no_op,
            prob_P1_no_op_dado_S1_no_op_S2_no_op_S3_op,
            prob_P1_no_op_dado_S1_no_op_S2_no_op_S3_no_op
        ]
    ])
    # Agregar la matriz a la lista de resultados de las réplicas
    matrices_probabilidades_replicas_P1.append(matriz_probabilidades_P1)

    #*****************************************************************
** Para P2
    # Inicializa los contadores de eventos relevantes para el calculo de
probabilidades condicionales de cada réplica para P2
    total_S1_op_S2_op_S3_op = 0
    total_S1_op_S2_op_S3_no_op = 0
    total_S1_op_S2_no_op_S3_op = 0
    total_S1_op_S2_no_op_S3_no_op= 0
    total_S1_no_op_S2_op_S3_op = 0
    total_S1_no_op_S2_op_S3_no_op = 0
    total_S1_no_op_S2_no_op_S3_op = 0
    total_S1_no_op_S2_no_op_S3_no_op = 0

    total_S1_op_S2_op_S3_op_P2_op = 0
    total_S1_op_S2_op_S3_no_op_P2_op = 0
    total_S1_op_S2_no_op_S3_op_P2_op = 0
    total_S1_op_S2_no_op_S3_no_op_P2_op = 0
    total_S1_no_op_S2_op_S3_op_P2_op = 0
    total_S1_no_op_S2_op_S3_no_op_P2_op = 0
    total_S1_no_op_S2_no_op_S3_op_P2_op = 0
    total_S1_no_op_S2_no_op_S3_no_op_P2_op = 0

    # Iterar a través de los días y contar eventos relevantes
    for i in range(num_dias):
        S1_op = 0 if not S1.historial_dias_sin_stock_provee[i] else 1
        S2_op = 0 if not S2.historial_dias_sin_stock_provee[i] else 1
        S3_op = 0 if not S3.historial_dias_sin_stock_provee[i] else 1
        P2_op = 0 if not P2.historial_dias_sin_stock_produ[i] else 1
```

```
        # Contar combinaciones de estados de S1, S2, S3
        if S1_op == 0 and S2_op == 0 and S3_op == 0:
            total_S1_op_S2_op_S3_op += 1
            if P2_op == 0:
                total_S1_op_S2_op_S3_op_P2_op += 1
        elif S1_op == 0 and S2_op == 0 and S3_op == 1:
            total_S1_op_S2_op_S3_no_op += 1
            if P2_op == 0:
                total_S1_op_S2_op_S3_no_op_P2_op += 1
        elif S1_op == 0 and S2_op == 1 and S3_op == 0:
            total_S1_op_S2_no_op_S3_op += 1
            if P2_op == 0:
                total_S1_op_S2_no_op_S3_op_P2_op += 1
        elif S1_op == 0 and S2_op == 1 and S3_op == 1:
            total_S1_op_S2_no_op_S3_no_op += 1
            if P2_op == 0:
                total_S1_op_S2_no_op_S3_no_op_P2_op += 1
        elif S1_op == 1 and S2_op == 0 and S3_op == 0:
            total_S1_no_op_S2_op_S3_op += 1
            if P2_op == 0:
                total_S1_no_op_S2_op_S3_op_P2_op += 1
        elif S1_op == 1 and S2_op == 0 and S3_op == 1:
            total_S1_no_op_S2_op_S3_no_op += 1
            if P2_op == 0:
                total_S1_no_op_S2_op_S3_no_op_P2_op += 1
        elif S1_op == 1 and S2_op == 1 and S3_op == 0:
            total_S1_no_op_S2_no_op_S3_op += 1
            if P2_op == 0:
                total_S1_no_op_S2_no_op_S3_op_P2_op += 1
        elif S1_op == 1 and S2_op == 1 and S3_op == 1:
            total_S1_no_op_S2_no_op_S3_no_op += 1
            if P2_op == 0:
                total_S1_no_op_S2_no_op_S3_no_op_P2_op += 1

    # Calcular probabilidades condicionales
    prob_P2_op_dado_S1_op_S2_op_S3_op = total_S1_op_S2_op_S3_op_P2_op /
total_S1_op_S2_op_S3_op if total_S1_op_S2_op_S3_op > 0 else 0.0
    prob_P2_op_dado_S1_op_S2_op_S3_no_op =
total_S1_op_S2_op_S3_no_op_P2_op / total_S1_op_S2_op_S3_no_op if
total_S1_op_S2_op_S3_no_op > 0 else 0.0
    prob_P2_op_dado_S1_op_S2_no_op_S3_op =
total_S1_op_S2_no_op_S3_op_P2_op / total_S1_op_S2_no_op_S3_op if
total_S1_op_S2_no_op_S3_op > 0 else 0.0
```

```
    prob_P2_op_dado_S1_op_S2_no_op_S3_no_op =
total_S1_op_S2_no_op_S3_no_op_P2_op / total_S1_op_S2_no_op_S3_no_op if
total_S1_op_S2_no_op_S3_no_op > 0 else 0.0
    prob_P2_op_dado_S1_no_op_S2_op_S3_op =
total_S1_no_op_S2_op_S3_op_P2_op / total_S1_no_op_S2_op_S3_op if
total_S1_no_op_S2_op_S3_op > 0 else 0.0
    prob_P2_op_dado_S1_no_op_S2_op_S3_no_op =
total_S1_no_op_S2_op_S3_no_op_P2_op / total_S1_no_op_S2_op_S3_no_op if
total_S1_no_op_S2_op_S3_no_op > 0 else 0.0
    prob_P2_op_dado_S1_no_op_S2_no_op_S3_op =
total_S1_no_op_S2_no_op_S3_op_P2_op / total_S1_no_op_S2_no_op_S3_op if
total_S1_no_op_S2_no_op_S3_op > 0 else 0.0
    prob_P2_op_dado_S1_no_op_S2_no_op_S3_no_op =
total_S1_no_op_S2_no_op_S3_no_op_P2_op / total_S1_no_op_S2_no_op_S3_no_op
if total_S1_no_op_S2_no_op_S3_no_op > 0 else 0.0

    # Calcular probabilidades complementarias
    prob_P2_no_op_dado_S1_op_S2_op_S3_op = 1 -
prob_P2_op_dado_S1_op_S2_op_S3_op
    prob_P2_no_op_dado_S1_op_S2_op_S3_no_op = 1 -
prob_P2_op_dado_S1_op_S2_op_S3_no_op
    prob_P2_no_op_dado_S1_op_S2_no_op_S3_op  = 1 -
prob_P2_op_dado_S1_op_S2_no_op_S3_op
    prob_P2_no_op_dado_S1_op_S2_no_op_S3_no_op = 1 -
prob_P2_op_dado_S1_op_S2_no_op_S3_no_op
    prob_P2_no_op_dado_S1_no_op_S2_op_S3_op = 1 -
prob_P2_op_dado_S1_no_op_S2_op_S3_op
    prob_P2_no_op_dado_S1_no_op_S2_op_S3_no_op = 1 -
prob_P2_op_dado_S1_no_op_S2_op_S3_no_op
    prob_P2_no_op_dado_S1_no_op_S2_no_op_S3_op = 1 -
prob_P2_op_dado_S1_no_op_S2_no_op_S3_op
    prob_P2_no_op_dado_S1_no_op_S2_no_op_S3_no_op = 1 -
prob_P2_op_dado_S1_no_op_S2_no_op_S3_no_op

    # Crear la matriz de probabilidades de P2 para la réplica
    # Primera fila: probabilidades de que P2 esté operativo
    # Segunda fila: probabilidades de que P2 no esté operativo
    matriz_probabilidades_P2 = np.array([
        [
            prob_P2_op_dado_S1_op_S2_op_S3_op,
            prob_P2_op_dado_S1_op_S2_op_S3_no_op,
            prob_P2_op_dado_S1_op_S2_no_op_S3_op,
            prob_P2_op_dado_S1_op_S2_no_op_S3_no_op,
            prob_P2_op_dado_S1_no_op_S2_op_S3_op,
            prob_P2_op_dado_S1_no_op_S2_op_S3_no_op,
```

```
            prob_P2_op_dado_S1_no_op_S2_no_op_S3_op,
            prob_P2_op_dado_S1_no_op_S2_no_op_S3_no_op
        ],
        [
            prob_P2_no_op_dado_S1_op_S2_op_S3_op,
            prob_P2_no_op_dado_S1_op_S2_op_S3_no_op,
            prob_P2_no_op_dado_S1_op_S2_no_op_S3_op,
            prob_P2_no_op_dado_S1_op_S2_no_op_S3_no_op,
            prob_P2_no_op_dado_S1_no_op_S2_op_S3_op,
            prob_P2_no_op_dado_S1_no_op_S2_op_S3_no_op,
            prob_P2_no_op_dado_S1_no_op_S2_no_op_S3_op,
            prob_P2_no_op_dado_S1_no_op_S2_no_op_S3_no_op
        ]
    ])
    # Agregar la matriz a la lista de resultados de las réplicas
    matrices_probabilidades_replicas_P2.append(matriz_probabilidades_P2)


    #*****************************************************************
** Para W1

    # Inicializa los contadores de eventos relevantes para el cálculo de
probabilidades condicionales de W1
    total_P1_op_P2_op = 0
    total_P1_op_P2_no_op = 0
    total_P1_no_op_P2_op = 0
    total_P1_no_op_P2_no_op = 0

    total_P1_op_P2_op_W1_op = 0
    total_P1_op_P2_no_op_W1_op = 0
    total_P1_no_op_P2_op_W1_op = 0
    total_P1_no_op_P2_no_op_W1_op  = 0

    # Iterar a través de los días y contar eventos relevantes
    for i in range(num_dias):
        P1_op = 0 if not P1.historial_dias_sin_stock_produ[i] else 1
        P2_op = 0 if not P2.historial_dias_sin_stock_produ[i] else 1
        W1_op = 0 if not W1.historial_dias_sin_stock_mayo[i] else 1

        # Contar combinaciones de estados de P1 y P2
        if P1_op == 0 and P2_op == 0:
            total_P1_op_P2_op += 1
            if W1_op == 0:
                total_P1_op_P2_op_W1_op += 1
        elif P1_op == 0 and P2_op == 1:
```

```
            total_P1_op_P2_no_op += 1
            if W1_op == 0:
                total_P1_op_P2_no_op_W1_op += 1
        elif P1_op == 1 and P2_op == 0:
            total_P1_no_op_P2_op += 1
            if W1_op == 0:
                total_P1_no_op_P2_op_W1_op += 1
        elif P1_op == 1 and P2_op == 1:
            total_P1_no_op_P2_no_op += 1
            if W1_op == 0:
                total_P1_no_op_P2_no_op_W1_op += 1

    # Calcular probabilidades condicionales
    prob_W1_op_dado_P1_op_P2_op = total_P1_op_P2_op_W1_op /
total_P1_op_P2_op if total_P1_op_P2_op > 0 else 0.0
    prob_W1_op_dado_P1_op_P2_no_op = total_P1_op_P2_no_op_W1_op /
total_P1_op_P2_no_op if total_P1_op_P2_no_op > 0 else 0.0
    prob_W1_op_dado_P1_no_op_P2_op = total_P1_no_op_P2_op_W1_op /
total_P1_no_op_P2_op if total_P1_no_op_P2_op > 0 else 0.0
    prob_W1_op_dado_P1_no_op_P2_no_op = total_P1_no_op_P2_no_op_W1_op /
total_P1_no_op_P2_no_op if total_P1_no_op_P2_no_op > 0 else 0.0

    # Calcular probabilidades complementarias
    prob_W1_no_op_dado_P1_op_P2_op = 1 - prob_W1_op_dado_P1_op_P2_op
    prob_W1_no_op_dado_P1_op_P2_no_op = 1 - prob_W1_op_dado_P1_op_P2_no_op
    prob_W1_no_op_dado_P1_no_op_P2_op = 1 - prob_W1_op_dado_P1_no_op_P2_op
    prob_W1_no_op_dado_P1_no_op_P2_no_op = 1 -
prob_W1_op_dado_P1_no_op_P2_no_op

    # Crear la matriz de probabilidades de W1 para la réplica
    # Primera fila: probabilidades de que W1 esté operativo
    # Segunda fila: probabilidades de que W1 no esté operativo
    matriz_probabilidades_W1 = np.array([
        [
            prob_W1_op_dado_P1_op_P2_op,
            prob_W1_op_dado_P1_op_P2_no_op,
            prob_W1_op_dado_P1_no_op_P2_op,
            prob_W1_op_dado_P1_no_op_P2_no_op
        ],
        [
            prob_W1_no_op_dado_P1_op_P2_op,
            prob_W1_no_op_dado_P1_op_P2_no_op,
            prob_W1_no_op_dado_P1_no_op_P2_op,
            prob_W1_no_op_dado_P1_no_op_P2_no_op
        ]
```

```
    ])
    # Agregar la matriz a la lista de resultados de las réplicas
    matrices_probabilidades_replicas_W1.append(matriz_probabilidades_W1)

    #*****************************************************************
** Para W2

    # Inicializa los contadores de eventos relevantes para el cálculo de
probabilidades condicionales de W2
    total_P1_op_P2_op = 0
    total_P1_op_P2_no_op = 0
    total_P1_no_op_P2_op = 0
    total_P1_no_op_P2_no_op = 0

    total_P1_op_P2_op_W2_op = 0
    total_P1_op_P2_no_op_W2_op = 0
    total_P1_no_op_P2_op_W2_op = 0
    total_P1_no_op_P2_no_op_W2_op  = 0

    # Iterar a través de los días y contar eventos relevantes
    for i in range(num_dias):
        P1_op = 0 if not P1.historial_dias_sin_stock_produ[i] else 1
        P2_op = 0 if not P2.historial_dias_sin_stock_produ[i] else 1
        W2_op = 0 if not W2.historial_dias_sin_stock_mayo[i] else 1

        # Contar combinaciones de estados de P1 y P2
        if P1_op == 0 and P2_op == 0:
            total_P1_op_P2_op += 1
            if W2_op == 0:
                total_P1_op_P2_op_W2_op += 1
        elif P1_op == 0 and P2_op == 1:
            total_P1_op_P2_no_op += 1
            if W2_op == 0:
                total_P1_op_P2_no_op_W2_op += 1
        elif P1_op == 1 and P2_op == 0:
            total_P1_no_op_P2_op += 1
            if W2_op == 0:
                total_P1_no_op_P2_op_W2_op += 1
        elif P1_op == 1 and P2_op == 1:
            total_P1_no_op_P2_no_op += 1
            if W2_op == 0:
                total_P1_no_op_P2_no_op_W2_op += 1

    # Calcular probabilidades condicionales
```

```
    prob_W2_op_dado_P1_op_P2_op = total_P1_op_P2_op_W2_op /
total_P1_op_P2_op if total_P1_op_P2_op > 0 else 0.0
    prob_W2_op_dado_P1_op_P2_no_op = total_P1_op_P2_no_op_W2_op /
total_P1_op_P2_no_op if total_P1_op_P2_no_op > 0 else 0.0
    prob_W2_op_dado_P1_no_op_P2_op = total_P1_no_op_P2_op_W2_op /
total_P1_no_op_P2_op if total_P1_no_op_P2_op > 0 else 0.0
    prob_W2_op_dado_P1_no_op_P2_no_op = total_P1_no_op_P2_no_op_W2_op /
total_P1_no_op_P2_no_op if total_P1_no_op_P2_no_op > 0 else 0.0

    # Calcular probabilidades complementarias
    prob_W2_no_op_dado_P1_op_P2_op = 1 - prob_W2_op_dado_P1_op_P2_op
    prob_W2_no_op_dado_P1_op_P2_no_op = 1 - prob_W2_op_dado_P1_op_P2_no_op
    prob_W2_no_op_dado_P1_no_op_P2_op = 1 - prob_W2_op_dado_P1_no_op_P2_op
    prob_W2_no_op_dado_P1_no_op_P2_no_op = 1 -
prob_W2_op_dado_P1_no_op_P2_no_op

    # Crear la matriz de probabilidades de W2 para la réplica
    # Primera fila: probabilidades de que W2 esté operativo
    # Segunda fila: probabilidades de que W2 no esté operativo
    matriz_probabilidades_W2 = np.array([
        [
            prob_W2_op_dado_P1_op_P2_op,
            prob_W2_op_dado_P1_op_P2_no_op,
            prob_W2_op_dado_P1_no_op_P2_op,
            prob_W2_op_dado_P1_no_op_P2_no_op
        ],
        [
            prob_W2_no_op_dado_P1_op_P2_op,
            prob_W2_no_op_dado_P1_op_P2_no_op,
            prob_W2_no_op_dado_P1_no_op_P2_op,
            prob_W2_no_op_dado_P1_no_op_P2_no_op
        ]
    ])
    # Agregar la matriz a la lista de resultados de las réplicas
    matrices_probabilidades_replicas_W2.append(matriz_probabilidades_W2)

    #*****************************************************************
** Para R1
    # Inicializa los contadores de eventos relevantes para el cálculo de
probabilidades condicionales de cada réplica para R1
    total_W1_op = 0
    total_W1_no_op = 0

    total_W1_op_R1_op = 0
    total_W1_no_op_R1_op = 0
```

```
    # Iterar a través de los días y contar eventos relevantes
    for i in range(num_dias):
        W1_op = 0 if not W1.historial_dias_sin_stock_mayo[i] else 1
        R1_op = 0 if not R1.historial_dias_sin_stock_mino[i] else 1

        # Contar combinaciones de estados de W1
        if W1_op == 0:
            total_W1_op += 1
            if R1_op == 0:
                total_W1_op_R1_op += 1
        elif W1_op == 1:
            total_W1_no_op += 1
            if R1_op == 0:
                total_W1_no_op_R1_op += 1

    # Calcular probabilidades condicionales
    prob_R1_op_dado_W1_op = total_W1_op_R1_op / total_W1_op if total_W1_op
> 0 else 0.0
    prob_R1_op_dado_W1_no_op = total_W1_no_op_R1_op / total_W1_no_op if
total_W1_no_op > 0 else 0.0

    # Calcular probabilidades complementarias
    prob_R1_no_op_dado_W1_op = 1 - prob_R1_op_dado_W1_op
    prob_R1_no_op_dado_W1_no_op = 1 - prob_R1_op_dado_W1_no_op

    # Crear la matriz de probabilidades de R1 para la réplica
    # Primera fila: probabilidades de que R1 esté operativo
    # Segunda fila: probabilidades de que R1 no esté operativo
    matriz_probabilidades_R1 = np.array([
        [
            prob_R1_op_dado_W1_op,
            prob_R1_op_dado_W1_no_op
        ],
        [
            prob_R1_no_op_dado_W1_op,
            prob_R1_no_op_dado_W1_no_op
        ]
    ])
    # Agregar la matriz a la lista de resultados de las réplicas
    matrices_probabilidades_replicas_R1.append(matriz_probabilidades_R1)


    #*****************************************************************
** Para R2
```

```
    # Inicializa los contadores de eventos relevantes para el cálculo de
probabilidades condicionales de cada réplica para R2
    total_W2_op = 0
    total_W2_no_op = 0

    total_W2_op_R2_op = 0
    total_W2_no_op_R2_op = 0

    # Iterar a través de los días y contar eventos relevantes para R2
    for i in range(num_dias):
        W2_op = 0 if not W2.historial_dias_sin_stock_mayo[i] else 1
        R2_op = 0 if not R2.historial_dias_sin_stock_mino[i] else 1

        # Contar combinaciones de estados de W2
        if W2_op == 0:
            total_W2_op += 1
            if R2_op == 0:
                total_W2_op_R2_op += 1
        elif W2_op == 1:
            total_W2_no_op += 1
            if R2_op == 0:
                total_W2_no_op_R2_op += 1

    # Calcular probabilidades condicionales para R2
    prob_R2_op_dado_W2_op = total_W2_op_R2_op / total_W2_op if total_W2_op
> 0 else 0.0
    prob_R2_op_dado_W2_no_op = total_W2_no_op_R2_op / total_W2_no_op if
total_W2_no_op > 0 else 0.0

    # Calcular probabilidades complementarias para R2
    prob_R2_no_op_dado_W2_op = 1 - prob_R2_op_dado_W2_op
    prob_R2_no_op_dado_W2_no_op = 1 - prob_R2_op_dado_W2_no_op

    # Crear la matriz de probabilidades de R2 para la réplica
    # Primera fila: probabilidades de que R2 esté operativo
    # Segunda fila: probabilidades de que R2 no esté operativo
    matriz_probabilidades_R2 = np.array([
        [
            prob_R2_op_dado_W2_op,
            prob_R2_op_dado_W2_no_op
        ],
        [
            prob_R2_no_op_dado_W2_op,
            prob_R2_no_op_dado_W2_no_op
        ]
```

```
    ])
    # Agregar la matriz a la lista de resultados de las réplicas
    matrices_probabilidades_replicas_R2.append(matriz_probabilidades_R2)

    #*****************************************************************
** Para R3
    # Inicializa los contadores de eventos relevantes para el cálculo de
probabilidades condicionales de cada réplica para R3
    total_W2_op = 0
    total_W2_no_op = 0

    total_W2_op_R3_op = 0
    total_W2_no_op_R3_op = 0

    # Iterar a través de los días y contar eventos relevantes para R3
    for i in range(num_dias):
        W2_op = 0 if not W2.historial_dias_sin_stock_mayo[i] else 1
        R3_op = 0 if not R3.historial_dias_sin_stock_mino[i] else 1

        # Contar combinaciones de estados de W2
        if W2_op == 0:
            total_W2_op += 1
            if R3_op == 0:
                total_W2_op_R3_op += 1
        elif W2_op == 1:
            total_W2_no_op += 1
            if R3_op == 0:
                total_W2_no_op_R3_op += 1

    # Calcular probabilidades condicionales para R3
    prob_R3_op_dado_W2_op = total_W2_op_R3_op / total_W2_op if total_W2_op
> 0 else 0.0
    prob_R3_op_dado_W2_no_op = total_W2_no_op_R3_op / total_W2_no_op if
total_W2_no_op > 0 else 0.0

    # Calcular probabilidades complementarias para R3
    prob_R3_no_op_dado_W2_op = 1 - prob_R3_op_dado_W2_op
    prob_R3_no_op_dado_W2_no_op = 1 - prob_R3_op_dado_W2_no_op

    # Crear la matriz de probabilidades de R3 para la réplica
    # Primera fila: probabilidades de que R3 esté operativo
    # Segunda fila: probabilidades de que R3 no esté operativo
    matriz_probabilidades_R3 = np.array([
        [
            prob_R3_op_dado_W2_op,
```

```
            prob_R3_op_dado_W2_no_op
        ],
        [
            prob_R3_no_op_dado_W2_op,
            prob_R3_no_op_dado_W2_no_op
        ]
    ])
    # Agregar la matriz a la lista de resultados de las réplicas
    matrices_probabilidades_replicas_R3.append(matriz_probabilidades_R3)

# Convertir la lista de S1 a un array de NumPy
matrices_probabilidades_replicas_S1 =
np.array(matrices_probabilidades_replicas_S1)
# Cálculo del promedio de las matrices de probabilidades condicionales
para S1
promedio_probabilidades_S1 = np.mean(matrices_probabilidades_replicas_S1,
axis=0)
# Extracción de los valores promedio para S1
promedio_probabilidades_S1_operativo = promedio_probabilidades_S1[0]
promedio_probabilidades_S1_no_operativo = promedio_probabilidades_S1[1]
# Definición de los estados
estados = ["Promedio"]
# Creación de la tabla para S1
tabla_S1 = pd.DataFrame({
    "Estado": estados,
    "S1 Operativo": promedio_probabilidades_S1_operativo,
    "S1 No Operativo": promedio_probabilidades_S1_no_operativo
})
# Mostrar la tabla como matriz de 2x1
print("Matriz para S1:")
print(tabla_S1.set_index('Estado').T.to_string())

# Convertir la lista de S2 a un array de NumPy
matrices_probabilidades_replicas_S2 =
np.array(matrices_probabilidades_replicas_S2)
# Cálculo del promedio de las matrices de probabilidades condicionales
para S2
promedio_probabilidades_S2 = np.mean(matrices_probabilidades_replicas_S2,
axis=0)
# Extracción de los valores promedio para S2
promedio_probabilidades_S2_operativo = promedio_probabilidades_S2[0]
promedio_probabilidades_S2_no_operativo = promedio_probabilidades_S2[1]
# Definición de los estados
estados = ["Promedio"]
# Creación de las tablas para S2
```

```
tabla_S2 = pd.DataFrame({
    "Estado": estados,
    "S2 Operativo": promedio_probabilidades_S2_operativo,
    "S2 No Operativo": promedio_probabilidades_S2_no_operativo
})
# Mostrar la tabla como matriz de 2x1
print("Matriz para S2:")
print(tabla_S2.set_index('Estado').T.to_string())

# Convertir la lista de S3 a un array de NumPy
matrices_probabilidades_replicas_S3 =
np.array(matrices_probabilidades_replicas_S3)
# Cálculo del promedio de las matrices de probabilidades condicionales
para S3
promedio_probabilidades_S3 = np.mean(matrices_probabilidades_replicas_S3,
axis=0)
# Extracción de los valores promedio para S3
promedio_probabilidades_S3_operativo = promedio_probabilidades_S3[0]
promedio_probabilidades_S3_no_operativo = promedio_probabilidades_S3[1]
# Definición de los estados
estados = ["Promedio"]
# Creación de la tabla para S3
tabla_S3 = pd.DataFrame({
    "Estado": estados,
    "S3 Operativo": promedio_probabilidades_S3_operativo,
    "S3 No Operativo": promedio_probabilidades_S3_no_operativo
})
# Mostrar la tabla como matriz de 2x1
print("Matriz para S3:")
print(tabla_S3.set_index('Estado').T.to_string())


# Convertir la lista de P1 a un array de NumPy
matrices_probabilidades_replicas_P1 =
np.array(matrices_probabilidades_replicas_P1)
# Cálculo del promedio de las matrices de probabilidades condicionales
para P1
promedio_probabilidades_P1 = np.mean(matrices_probabilidades_replicas_P1,
axis=0)
# Extracción de los valores promedio para P1
promedio_probabilidades_P1_operativo = promedio_probabilidades_P1[0]
promedio_probabilidades_P1_no_operativo = promedio_probabilidades_P1[1]
# Definición de los estados
estados = ["(0,0,0)", "(0,0,1)", "(0,1,0)", "(0,1,1)", "(1,0,0)",
"(1,0,1)", "(1,1,0)", "(1,1,1)"]
```

```
# Creación de la tabla para P1
tabla_P1 = pd.DataFrame({
    "Estado": estados,
    "P1 Operativo": promedio_probabilidades_P1_operativo.ravel(),  # Usar
.ravel() para aplanar la matriz
    "P1 No Operativo": promedio_probabilidades_P1_no_operativo.ravel()  #
Usar .ravel() para aplanar la matriz
})
# Mostrar la tabla como matriz de 2x8
print("Matriz para P1:")
print(tabla_P1.set_index('Estado').T.to_string())

# Convertir la lista de P2 a un array de NumPy
matrices_probabilidades_replicas_P2 =
np.array(matrices_probabilidades_replicas_P2)
# Cálculo del promedio de las matrices de probabilidades condicionales
para P2
promedio_probabilidades_P2 = np.mean(matrices_probabilidades_replicas_P2,
axis=0)
# Extracción de los valores promedio para P2
promedio_probabilidades_P2_operativo = promedio_probabilidades_P2[0]
promedio_probabilidades_P2_no_operativo = promedio_probabilidades_P2[1]
# Definición de los estados
estados = ["(0,0,0)", "(0,0,1)", "(0,1,0)", "(0,1,1)", "(1,0,0)",
"(1,0,1)", "(1,1,0)", "(1,1,1)"]
# Creación de la tabla para P2
tabla_P2 = pd.DataFrame({
    "Estado": estados,
    "P2 Operativo": promedio_probabilidades_P2_operativo.ravel(),  # Usar
.ravel() para aplanar la matriz
    "P2 No Operativo": promedio_probabilidades_P2_no_operativo.ravel()  #
Usar .ravel() para aplanar la matriz
})
# Mostrar la tabla como matriz de 2x8
print("Matriz para P2:")
print(tabla_P2.set_index('Estado').T.to_string())


# Convertir la lista de W1 a un array de NumPy
matrices_probabilidades_replicas_W1 =
np.array(matrices_probabilidades_replicas_W1)
# Cálculo del promedio de las matrices de probabilidades condicionales
para W1
promedio_probabilidades_W1 = np.mean(matrices_probabilidades_replicas_W1,
axis=0)
```

```
# Extracción de los valores promedio para W1
promedio_probabilidades_W1_operativo = promedio_probabilidades_W1[0]
promedio_probabilidades_W1_no_operativo = promedio_probabilidades_W1[1]
# Definición de los estados
estados = ["(0,0)", "(0,1)", "(1,0)", "(1,1)"]
# Creación de la tabla para W1
tabla_W1 = pd.DataFrame({
    "Estado": estados,
    "W1 Operativo": promedio_probabilidades_W1_operativo.ravel(),  # Usar
.ravel() para aplanar la matriz
    "W1 No Operativo": promedio_probabilidades_W1_no_operativo.ravel()  #
Usar .ravel() para aplanar la matriz
})
# Mostrar la tabla como matriz de 2x4
print("Matriz para W1:")
print(tabla_W1.set_index('Estado').T.to_string())

# Convertir la lista de W2 a un array de NumPy
matrices_probabilidades_replicas_W2 =
np.array(matrices_probabilidades_replicas_W2)
# Cálculo del promedio de las matrices de probabilidades condicionales
para W2
promedio_probabilidades_W2 = np.mean(matrices_probabilidades_replicas_W2,
axis=0)
# Extracción de los valores promedio para W2
promedio_probabilidades_W2_operativo = promedio_probabilidades_W2[0]
promedio_probabilidades_W2_no_operativo = promedio_probabilidades_W2[1]
# Definición de los estados
estados = ["(0,0)", "(0,1)", "(1,0)", "(1,1)"]
# Creación de la tabla para W2
tabla_W2 = pd.DataFrame({
    "Estado": estados,
    "W2 Operativo": promedio_probabilidades_W2_operativo.ravel(),  # Usar
.ravel() para aplanar la matriz
    "W2 No Operativo": promedio_probabilidades_W2_no_operativo.ravel()  #
Usar .ravel() para aplanar la matriz
})
# Mostrar la tabla como matriz de 2x4
print("Matriz para W2:")
print(tabla_W2.set_index('Estado').T.to_string())


# Convertir la lista de R1 a un array de NumPy
matrices_probabilidades_replicas_R1 =
np.array(matrices_probabilidades_replicas_R1)
```

```
# Cálculo del promedio de las matrices de probabilidades condicionales
para R1
promedio_probabilidades_R1 = np.mean(matrices_probabilidades_replicas_R1,
axis=0)
# Extracción de los valores promedio para R1
promedio_probabilidades_R1_operativo = promedio_probabilidades_R1[0]
promedio_probabilidades_R1_no_operativo = promedio_probabilidades_R1[1]
# Definición de los estados para R1
estados_R1 = ["(0)", "(1)"]
# Creación de la tabla para R1
tabla_R1 = pd.DataFrame({
    "Estado R1": estados_R1,
    "R1 Operativo": promedio_probabilidades_R1_operativo.ravel(),  # Usar
.ravel() para aplanar la matriz
    "R1 No Operativo": promedio_probabilidades_R1_no_operativo.ravel()  #
Usar .ravel() para aplanar la matriz
})
# Mostrar la tabla como matriz de 2x2
print("Matriz para R1:")
print(tabla_R1.set_index('Estado R1').T.to_string())

# Convertir la lista de R2 a un array de NumPy
matrices_probabilidades_replicas_R2 =
np.array(matrices_probabilidades_replicas_R2)
# Cálculo del promedio de las matrices de probabilidades condicionales
para R2
promedio_probabilidades_R2 = np.mean(matrices_probabilidades_replicas_R2,
axis=0)
# Extracción de los valores promedio para R2
promedio_probabilidades_R2_operativo = promedio_probabilidades_R2[0]
promedio_probabilidades_R2_no_operativo = promedio_probabilidades_R2[1]
# Definición de los estados para R2
estados_R2 = ["(0)", "(1)"]
# Creación de la tabla para R2
tabla_R2 = pd.DataFrame({
    "Estado R2": estados_R2,
    "R2 Operativo": promedio_probabilidades_R2_operativo.ravel(),  # Usar
.ravel() para aplanar la matriz
    "R2 No Operativo": promedio_probabilidades_R2_no_operativo.ravel()  #
Usar .ravel() para aplanar la matriz
})
# Mostrar la tabla como matriz de 2x2
print("Matriz para R2:")
print(tabla_R2.set_index('Estado R2').T.to_string())
```

```
# Convertir la lista de R3 a un array de NumPy
matrices_probabilidades_replicas_R3 =
np.array(matrices_probabilidades_replicas_R3)
# Cálculo del promedio de las matrices de probabilidades condicionales
para R3
promedio_probabilidades_R3 = np.mean(matrices_probabilidades_replicas_R3,
axis=0)
# Extracción de los valores promedio para R3
promedio_probabilidades_R3_operativo = promedio_probabilidades_R3[0]
promedio_probabilidades_R3_no_operativo = promedio_probabilidades_R3[1]
# Definición de los estados para R3
estados_R3 = ["(0)", "(1)"]
# Creación de la tabla para R3
tabla_R3 = pd.DataFrame({
    "Estado R3": estados_R3,
    "R3 Operativo": promedio_probabilidades_R3_operativo.ravel(),
    "R3 No Operativo": promedio_probabilidades_R3_no_operativo.ravel()
})
# Mostrar la tabla como matriz de 2x2
print("Matriz para R3:")
print(tabla_R3.set_index('Estado R3').T.to_string())


# CREACIÓN DEL MODELO DE RED BAYESIANA
!pip install pgmpy
from pgmpy.models import BayesianNetwork   # Importa la clase
BayesianNetwork desde la biblioteca pgmpy.models
from pgmpy.factors.discrete import TabularCPD # Importa la clase
TabularCPD desde pgmpy.factors.discrete
from pgmpy.inference import VariableElimination # Importa la clase
VariableElimination desde pgmpy.inference

# Definición del Modelo de Red Bayesiana (Crear un objeto llamado 'modelo'
con la estructura de la Red)
modelo = BayesianNetwork([('S1', 'P1'), ('S2', 'P1'), ('S3', 'P1'),
                          ('S1', 'P2'), ('S2', 'P2'), ('S3', 'P2'),
                          ('P1', 'W1'), ('P1', 'W2'),
                          ('P2', 'W1'), ('P2', 'W2'),
                          ('W1', 'R1'),
                          ('W2', 'R2'), ('W2', 'R3')])

#Definición de las Tablas de Probabilidad Condicional (CPT)
# Definir la CPT para S1 con las probabilidades promedio
S1_cpd = TabularCPD(variable='S1',
```

```
                    variable_card=2,  # Cardinalidad de S1 (2 estados:
Operativo y No Operativo)
                    values=[promedio_probabilidades_S1_operativo,
promedio_probabilidades_S1_no_operativo])
# Definir la CPT para S2 con las probabilidades promedio
S2_cpd = TabularCPD(variable='S2',
                    variable_card=2,  # Cardinalidad de S2 (2 estados:
Operativo y No Operativo)
                    values=[promedio_probabilidades_S2_operativo,
promedio_probabilidades_S2_no_operativo])
# Definir la CPT para S3 con las probabilidades promedio
S3_cpd = TabularCPD(variable='S3',
                    variable_card=2,  # Cardinalidad de S3 (2 estados:
Operativo y No Operativo)
                    values=[promedio_probabilidades_S3_operativo,
promedio_probabilidades_S3_no_operativo])
# Definir la CPT para P1 con las probabilidades promedio
P1_cpd = TabularCPD(variable='P1', variable_card=2,
                    values=[promedio_probabilidades_P1_operativo.ravel(),
                            promedio_probabilidades_P1_no_operativo.ravel(
)],
                    evidence=['S1', 'S2', 'S3'], evidence_card=[2, 2, 2])
# Definir la CPT para P2 con las probabilidades promedio
P2_cpd = TabularCPD(variable='P2', variable_card=2,
                    values=[promedio_probabilidades_P2_operativo.ravel(),
                            promedio_probabilidades_P2_no_operativo.ravel(
)],
                    evidence=['S1', 'S2', 'S3'], evidence_card=[2, 2, 2])
# Definir la CPT para W1 con las probabilidades promedio
W1_cpd = TabularCPD(variable='W1',
                    variable_card=2,  # Cardinalidad de W1 (2 estados:
Operativo y No Operativo)
                    values=[promedio_probabilidades_W1_operativo.ravel(),
                            promedio_probabilidades_W1_no_operativo.ravel(
)],
                    evidence=['P1', 'P2'], evidence_card=[2, 2])
# Definir la CPT para W2 con las probabilidades promedio
W2_cpd = TabularCPD(variable='W2',
                    variable_card=2,  # Cardinalidad de W2 (2 estados:
Operativo y No Operativo)
                    values=[promedio_probabilidades_W2_operativo.ravel(),
                            promedio_probabilidades_W2_no_operativo.ravel(
)],
                    evidence=['P1', 'P2'], evidence_card=[2, 2])
# Definir la CPT para R1 con las probabilidades promedio
```

```
R1_cpd = TabularCPD(variable='R1',
                    variable_card=2,  # Cardinalidad de R1 (2 estados:
Operativo y No Operativo)
                    values=[promedio_probabilidades_R1_operativo.ravel(),
                            promedio_probabilidades_R1_no_operativo.ravel(
)],
                    evidence=['W1'], evidence_card=[2])  # Indicando que
depende de W1 que tiene 2 estados
# Definir la CPT para R2 con las probabilidades promedio
R2_cpd = TabularCPD(variable='R2',
                    variable_card=2,  # Cardinalidad de R2 (2 estados:
Operativo y No Operativo)
                    values=[promedio_probabilidades_R2_operativo.ravel(),
                            promedio_probabilidades_R2_no_operativo.ravel(
)],
                    evidence=['W2'], evidence_card=[2])  # Indicando que
depende de W2 que tiene 2 estados
# Definir la CPT para R2 con las probabilidades promedio
R3_cpd = TabularCPD(variable='R3',
                    variable_card=2,  # Cardinalidad de R3 (2 estados:
Operativo y No Operativo)
                    values=[promedio_probabilidades_R3_operativo.ravel(),
                            promedio_probabilidades_R3_no_operativo.ravel(
)],
                    evidence=['W2'], evidence_card=[2])  # Indicando que
depende de W2 que tiene 2 estado

# Asignación de las CPT a los Nodos del Modelo
modelo.add_cpds(S1_cpd, S2_cpd, S3_cpd, P1_cpd, P2_cpd, W1_cpd, W2_cpd,
R1_cpd, R2_cpd, R3_cpd)

# Verificación del modelo
assert modelo.check_model()


#VISUALIZACIÓN DEL MODELO DE RED BAYESIANA

!pip install --upgrade pgmpy networkx matplotlib
import networkx as nx
import matplotlib.pyplot as plt

# Diccionario del modelo de red bayesiana para visualización (grafo)
modelo_dict = {
    'red': {
```

```
        'nodos': ['S1', 'S2', 'S3', 'P1', 'P2', 'W1', 'W2', 'R1', 'R2',
'R3'],
        'arcos': [
            ('S1', 'P1'), ('S2', 'P1'), ('S3', 'P1'),
            ('S1', 'P2'), ('S2', 'P2'), ('S3', 'P2'),
            ('P1', 'W1'), ('P1', 'W2'),
            ('P2', 'W1'), ('P2', 'W2'),
            ('W1', 'R1'),
            ('W2', 'R2'), ('W2', 'R3')
        ]
    },
    'tablas_probabilidad_condicional': {
        'S1': {
            'cardinalidad': 2,
            'valores': [promedio_probabilidades_S1_operativo,
promedio_probabilidades_S1_no_operativo]
        },
        'S2': {
            'cardinalidad': 2,
            'valores': [promedio_probabilidades_S2_operativo,
promedio_probabilidades_S2_no_operativo]
        },
        'S3': {
            'cardinalidad': 2,
            'valores': [promedio_probabilidades_S3_operativo,
promedio_probabilidades_S3_no_operativo]
        },
        'P1': {
            'cardinalidad': 2,
            'valores': [promedio_probabilidades_P1_operativo.ravel(),
promedio_probabilidades_P1_no_operativo.ravel()],
            'evidencia': ['S1', 'S2', 'S3'],
            'cardinalidad_evidencia': [2, 2, 2]
        },
        'P2': {
            'cardinalidad': 2,
            'valores': [promedio_probabilidades_P2_operativo.ravel(),
promedio_probabilidades_P2_no_operativo.ravel()],
            'evidencia': ['S1', 'S2', 'S3'],
            'cardinalidad_evidencia': [2, 2, 2]
        },
        'W1': {
            'cardinalidad': 2,
            'valores': [promedio_probabilidades_W1_operativo.ravel(),
promedio_probabilidades_W1_no_operativo.ravel()],
```

```
            'evidencia': ['P1', 'P2'],
            'cardinalidad_evidencia': [2, 2]
        },
        'W2': {
            'cardinalidad': 2,
            'valores': [promedio_probabilidades_W2_operativo.ravel(),
promedio_probabilidades_W2_no_operativo.ravel()],
            'evidencia': ['P1', 'P2'],
            'cardinalidad_evidencia': [2, 2]
        },
        'R1': {
            'cardinalidad': 2,
            'valores': [promedio_probabilidades_R1_operativo.ravel(),
promedio_probabilidades_R1_no_operativo.ravel()],
            'evidencia': ['W1'],
            'cardinalidad_evidencia': [2]
        },
        'R2': {
            'cardinalidad': 2,
            'valores': [promedio_probabilidades_R2_operativo.ravel(),
promedio_probabilidades_R2_no_operativo.ravel()],
            'evidencia': ['W2'],
            'cardinalidad_evidencia': [2]
        },
        'R3': {
            'cardinalidad': 2,
            'valores': [promedio_probabilidades_R3_operativo.ravel(),
promedio_probabilidades_R3_no_operativo.ravel()],
            'evidencia': ['W2'],
            'cardinalidad_evidencia': [2]
        }
    }
}

# Definir etiquetas para las probabilidades
proba = {
    'S1': f'Operativo: {promedio_probabilidades_S1_operativo}\nNo
Operativo: {promedio_probabilidades_S1_no_operativo}',
    'S2': f'Operativo: {promedio_probabilidades_S2_operativo}\nNo
Operativo: {promedio_probabilidades_S2_no_operativo}',
    'S3': f'Operativo: {promedio_probabilidades_S3_operativo}\nNo
Operativo: {promedio_probabilidades_S3_no_operativo}',
    'P1': f'Operativo: {promedio_probabilidades_P1_operativo.ravel()}\nNo
Operativo: {promedio_probabilidades_P1_no_operativo.ravel()}',
```

```
    'P2': f'Operativo: {promedio_probabilidades_P2_operativo.ravel()}\nNo
Operativo: {promedio_probabilidades_P2_no_operativo.ravel()}',
    'W1': f'Operativo: {promedio_probabilidades_W1_operativo.ravel()}\nNo
Operativo: {promedio_probabilidades_W1_no_operativo.ravel()}',
    'W2': f'Operativo: {promedio_probabilidades_W2_operativo.ravel()}\nNo
Operativo: {promedio_probabilidades_W2_no_operativo.ravel()}',
    'R1': f'Operativo: {promedio_probabilidades_R1_operativo.ravel()}\nNo
Operativo: {promedio_probabilidades_R1_no_operativo.ravel()}',
    'R2': f'Operativo: {promedio_probabilidades_R2_operativo.ravel()}\nNo
Operativo: {promedio_probabilidades_R2_no_operativo.ravel()}',
    'R3': f'Operativo: {promedio_probabilidades_R3_operativo.ravel()}\nNo
Operativo: {promedio_probabilidades_R3_no_operativo.ravel()}',
}

# Obtener las relaciones del modelo (arcos entre nodos)
edges = modelo.edges()
# Crear un grafo de NetworkX a partir de los arcos
G = nx.DiGraph()  # Crear un grafo dirigido
G.add_edges_from(edges)  # Añadir los arcos

# Aumentar el tamaño de la figura
plt.figure(figsize=(10, 5))  # Aumenta el tamaño de la figura
# Cambiar el layout y ajustar la disposición
pos = nx.spring_layout(G, k=1.8)  # Ajusta el parámetro 'k' para espaciar
más los nodos
# Dibujar el grafo
nx.draw(G, pos, with_labels=True, node_size=1000, node_color='lightblue',
font_size=10, font_weight='bold')
# Añadir etiquetas con un pequeño offset para evitar superposición
labels = {node: proba[node] for node in G.nodes()}
offset = 0.1  # Ajustar el offset según sea necesario
# Dibujar las etiquetas
for node, (x, y) in pos.items():
    plt.text(x, y - offset, s=labels[node], fontsize=7, ha='center',
va='top')

# Mostrar el grafo
plt.title("**********MODELO DE RED BAYESIANA**********")
plt.show()


# OBJETO INFERENCIAS PARA ANÁLISIS DE RIESGOS Y RESILIENCIA
# Crear el objeto "inferencia" utilizando el algoritmo de eliminación de
variables
inferencia = VariableElimination(modelo)
```

```
print(f"{negrita}**********************************************{fin_negrita}")
# ANÁLISIS DE RIESGOS
print(f"ANÁLISIS DE RIESGOS")
print(f"{negrita}**********************************************{fin_negrita}")
# Lista de TODOS los participantes
lista_todos_los_participantes = ['S1', 'S2', 'S3', 'P1', 'P2', 'W1', 'W2', 'R1', 'R2', 'R3']

# Realizar inferencias e imprimir las probabilidades marginales para cada participante
# Probabilidad de que cada participante esté operativo o no
print("Sin evidencia:")
for participante in lista_todos_los_participantes:
    probabilidad_total_sin_evidencia = inferencia.query(variables=[participante])
    print(f"P({participante}=0) (Operativo) =", probabilidad_total_sin_evidencia.values[0])
    print(f"P({participante}=1) (No operativo) =", probabilidad_total_sin_evidencia.values[1])

print(f"{negrita}**********************************************{fin_negrita}")
# ANÁLISIS DE RESILIENCIA
print(f"ANÁLISIS DE RESILIENCIA")
print(f"{negrita}**********************************************{fin_negrita}")
# Lista de participantes para actualizar evidencias
participantesR1 = ['S1', 'S2', 'S3', 'P1', 'P2', 'W1', 'W2', 'R2', 'R3']
participantesR2 = ['S1', 'S2', 'S3', 'P1', 'P2', 'W1', 'W2', 'R1', 'R3']
participantesR3 = ['S1', 'S2', 'S3', 'P1', 'P2', 'W1', 'W2', 'R1', 'R2']

# Se define la siguiente funcion para calcular la resiliencia con (B_O, B_O_X_bar_i)
def calcular_resiliencia(variable_R, participantes):
    # Probabilidad base de que R esté en estado operativo
    probabilidad_base = inferencia.query(variables=[variable_R])
    B_O = probabilidad_base.values[0]  # Operativo es el estado 0

    resultados_resiliencia = []

    for participante in participantes:
```

```
        # Probabilidad de que R esté operativo cuando el participante está
en estado no operativo
        evidencia_no_operativo = inferencia.query(variables=[variable_R],
evidence={participante: 1})
        B_O_X_bar_i = evidencia_no_operativo.values[0]

        # Cálculo de resiliencia
        resiliencia_no_operativo = 1 + ((B_O_X_bar_i - B_O) / B_O)

        # Añadir resultados al DataFrame
        resultados_resiliencia.append({
            'Participante': participante,
            'B_O': B_O,
            'B_O_X_bar_i': B_O_X_bar_i,
            'Resiliencia No Operativo': resiliencia_no_operativo,
        })

    return pd.DataFrame(resultados_resiliencia)

# Ajustar la configuración de visualización
pd.set_option('display.max_columns', None)  # Muestra todas las columnas
pd.set_option('display.max_colwidth', None)  # Muestra el contenido
completo de cada columna
pd.set_option('display.width', 1000)         # Ajusta el ancho total de la
visualización
pd.set_option('display.max_rows', None)      # Muestra todas las filas (si
es necesario)

# Cálculos para R1
tabla_R1 = calcular_resiliencia('R1', participantesR1)
print("Tabla para R1")
print(tabla_R1.to_string(index=False))

# Cálculos para R2
tabla_R2 = calcular_resiliencia('R2', participantesR2)
print("Tabla para R2")
print(tabla_R2.to_string(index=False))

# Cálculos para R3
tabla_R3 = calcular_resiliencia('R3', participantesR3)
print("Tabla para R3")
print(tabla_R3.to_string(index=False))

print(f"{negrita}**********************************************{fin_negr
ita}")
```

```
# ANÁLISIS DE SENSIBILIDAD
print(f"ANÁLISIS DE SENSIBILIDAD")
print(f"{negrita}**********************************************{fin_negrita}")
# Estados para introducir evidencia
estados = [0, 1]  # Dos posibles estados
# Lista de variables a analizar y los correspondientes Minoristas (R)
variables_a_analizar = {
    'S1': ['R1', 'R2', 'R3'],
    'S2': ['R1', 'R2', 'R3'],
    'S3': ['R1', 'R2', 'R3'],
    'P1': ['R1', 'R2', 'R3'],
    'P2': ['R1', 'R2', 'R3'],
    'W1': ['R1'],
    'W2': ['R2', 'R3']
}

def sensibilidad_variable(modelo, variable, estados, variables_R):
    sensibilidad = []
    for estado in estados:
        # Establecer el valor de la variable (0 para Operativo, 1 para No Operativo)
        evidencia = {variable: estado}  # Asignar el estado

        # Realizar inferencia
        inferencia = VariableElimination(modelo)

        # Calcular probabilidades para las variables R especificadas
        probabilidades = []
        for R in variables_R:
            prob_con_evidencia = inferencia.query(variables=[R], evidence=evidencia)
            probabilidades.append((prob_con_evidencia.values[0], prob_con_evidencia.values[1]))  # (Operativo, No Operativo)

        sensibilidad.append(probabilidades)

        # Imprimir las probabilidades  Con evidencia:  Estableciedno que cada variable que influye en el respectivo minorista
        # está "Operativo" o "Operativo" y calcular la probabilidad de minorista
        print(f"\nEvidencia: {variable} {'Operativo' if estado == 0 else 'No Operativo'}")
        for i, R in enumerate(variables_R):
            print(f"P({R}=0) (Operativo) =", prob_con_evidencia.values[0])
```

```
            print(f"P({R}=1) (No operativo) =",
prob_con_evidencia.values[1])

    return sensibilidad

# Función para organizar y graficar (Operativo y No Operativo por
separado)
def grafico_diferencias(resultados_totales, variables_a_analizar):
    difs_operativo = {}  # Para almacenar las diferencias de R=0
    difs_no_operativo = {}  # Para almacenar las diferencias de R=1

    for var, resultados in resultados_totales.items():
        resultados = np.array(resultados)
        for i, R in enumerate(variables_a_analizar[var]):
            # Calcular las diferencias
            diferencia_operativo = resultados[0, i, 0] - resultados[1, i,
0]  # Para R=0 (Operativo)
            diferencia_no_operativo = resultados[0, i, 1] - resultados[1,
i, 1]  # Para R=1 (No Operativo)

            # Almacenar las diferencias en los diccionarios
correspondientes
            difs_operativo[f'{var} - {R}'] = diferencia_operativo
            difs_no_operativo[f'{var} - {R}'] = diferencia_no_operativo

    # Ordenar las variables por el impacto de las diferencias (de mayor a
menor)
    sorted_var_operativo = sorted(difs_operativo.items(), key=lambda x:
abs(x[1]), reverse=True)
    sorted_var_no_operativo = sorted(difs_no_operativo.items(), key=lambda
x: abs(x[1]), reverse=True)

    # Datos para los gráficos
    etiquetas_operativo = [item[0] for item in sorted_var_operativo]
    difs_sorted_operativo = [item[1] for item in sorted_var_operativo]

    etiquetas_no_operativo = [item[0] for item in sorted_var_no_operativo]
    difs_sorted_no_operativo = [item[1] for item in
sorted_var_no_operativo]

    # Crear gráfico diferencias (R=0) Operativo
    plt.figure(figsize=(8, 5))
    bars_operativo = plt.barh(etiquetas_operativo, difs_sorted_operativo,
color='skyblue')
    plt.xlabel('Diferencias de Probabilidades Operativo (R=0)')
```

```
    plt.title('Gráfico del Análisis de Sensibilidad - Operativo')
    plt.grid(True)
    # Agregar etiquetas de valores en las barras
    for bar in bars_operativo:
        plt.text(bar.get_width(), bar.get_y() + bar.get_height()/2,
f'{bar.get_width():.6f}',
                 va='center', ha='left', color='black', fontsize=8)
    plt.show()

    # Crear gráfico diferencias (R=1) No Operativo
    plt.figure(figsize=(8, 5))
    bars_no_operativo = plt.barh(etiquetas_no_operativo,
difs_sorted_no_operativo, color='lightcoral')
    plt.xlabel('Diferencias de Probabilidades No Operativo (R=1)')
    plt.title('Gráfico del Análisis de Sensibilidad - No Operativo')
    plt.grid(True)
    # Agregar etiquetas de valores en las barras
    for bar in bars_no_operativo:
        plt.text(bar.get_width(), bar.get_y() + bar.get_height()/2,
f'{bar.get_width():.6f}',
                 va='center', ha='left', color='black', fontsize=8)
    plt.show()

# Resultados para cada variable
resultados_totales = {}
for var, variables_R in variables_a_analizar.items():
    resultados_totales[var] = sensibilidad_variable(modelo, var, estados,
variables_R)

# Generar los gráficos
grafico_diferencias(resultados_totales, variables_a_analizar)
```

Apéndice E. Código de Programación - Validación de Probabilidades Marginales

En el entorno de desarrollo de Google Colab, utilizando Python, el código es el siguiente:

```
import numpy as np

# Definir las probabilidades
probabilidades = {
    0: np.array([[0.984333], [0.015667]]),
    1: np.array([[0.988778], [0.011222]]),
    2: np.array([[0.992], [0.008]]),
    3: np.array([[0.01546, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0],
                 [0.98454, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0]]),
    4: np.array([[0.2585, 0.0, 0.075649, 0.0, 0.043258, 0.1, 0.1, 0.0],
                 [0.7415, 1.0, 0.924351, 1.0, 0.956742, 0.9, 0.9, 1.0]]),
    5: np.array([[0.969424, 0.726573, 0.754836, 0.663411],
                 [0.030576, 0.273427, 0.245164, 0.336589]]),
    6: np.array([[0.860652, 0.578601, 0.546523, 0.399211],
                 [0.139348, 0.421399, 0.453477, 0.600789]]),
    7: np.array([[0.978231, 0.994352],
                 [0.021769, 0.005648]]),
    8: np.array([[0.954937, 0.973015],
                 [0.045063, 0.026985]]),
    9: np.array([[0.913718, 0.955815],
                 [0.086282, 0.044185]])
}


# Obtener la probabilidad marginal de S1
def calcular_probabilidad_S1():
    return probabilidades[0]

print(f"P(S1 = 0): [{probabilidades[0][0, 0]}]")
print(f"P(S1 = 1): [{probabilidades[0][1, 0]}]")

# Obtener la probabilidad marginal de S2
def calcular_probabilidad_S2():
    return probabilidades[1]

print(f"P(S2 = 0): [{probabilidades[1][0, 0]}]")
print(f"P(S2 = 1): [{probabilidades[1][1, 0]}]")

# Obtener la probabilidad marginal de S3
def calcular_probabilidad_S3():
```

```
    return probabilidades[2]

print(f"P(S3 = 0): [{probabilidades[2][0, 0]}]")
print(f"P(S3 = 1): [{probabilidades[2][1, 0]}]")


# Inicializar las probabilidades marginales para P1
P_P1_0 = 0  # Para P1 = 0
P_P1_1 = 0  # Para P1 = 1

# Iterar sobre todas las combinaciones posibles de S1, S2, S3
for S1 in range(2):
    for S2 in range(2):
        for S3 in range(2):
            # Calcular las probabilidades condicionales para P1
            prob_P1_dado_S1_S2_S3_0 = probabilidades[3][0, S1 * 4 + S2 * 2
+ S3]
            prob_P1_dado_S1_S2_S3_1 = probabilidades[3][1, S1 * 4 + S2 * 2
+ S3]

            # Calcular las probabilidades marginales de S1, S2 y S3
            prob_S1 = probabilidades[0][S1]
            prob_S2 = probabilidades[1][S2]
            prob_S3 = probabilidades[2][S3]

            # Sumar las probabilidades
            P_P1_0 += prob_P1_dado_S1_S2_S3_0 * prob_S1 * prob_S2 *
prob_S3
            P_P1_1 += prob_P1_dado_S1_S2_S3_1 * prob_S1 * prob_S2 *
prob_S3

# Mostrar los resultados
print("P(P1 = 0):", P_P1_0)
print("P(P1 = 1):", P_P1_1)


# Inicializar las probabilidades marginales para P2
P_P2_0 = 0  # Para P2 = 0
P_P2_1 = 0  # Para P2 = 1

# Iterar sobre todas las combinaciones posibles de S1, S2, S3
for S1 in range(2):
    for S2 in range(2):
        for S3 in range(2):
            # Calcular las probabilidades condicionales para P2
```

```
            prob_P2_dado_S1_S2_S3_0 = probabilidades[4][0, S1 * 4 + S2 * 2
+ S3]
            prob_P2_dado_S1_S2_S3_1 = probabilidades[4][1, S1 * 4 + S2 * 2
+ S3]

            # Calcular las probabilidades marginales de S1, S2 y S3
            prob_S1 = probabilidades[0][S1]
            prob_S2 = probabilidades[1][S2]
            prob_S3 = probabilidades[2][S3]

            # Sumar las probabilidades
            P_P2_0 += prob_P2_dado_S1_S2_S3_0 * prob_S1 * prob_S2 *
prob_S3
            P_P2_1 += prob_P2_dado_S1_S2_S3_1 * prob_S1 * prob_S2 *
prob_S3

# Mostrar los resultados
print("P(P2 = 0):", P_P2_0)
print("P(P2 = 1):", P_P2_1)


# Inicializar las probabilidades marginales para W1
P_W1_0 = 0  # Para W1 = 0
P_W1_1 = 0  # Para W1 = 1

# Iterar sobre todas las combinaciones posibles de S1, S2, S3, P1 y P2
for S1 in range(2):
    for S2 in range(2):
        for S3 in range(2):
            for P1 in range(2):
                for P2 in range(2):
                    # Calcular las probabilidades condicionales para W1
                    prob_W1_dado_P1_P2_0 = probabilidades[5][0, P1 * 2 +
P2]
                    prob_W1_dado_P1_P2_1 = probabilidades[5][1, P1 * 2 +
P2]

                    # Calcular las probabilidades marginales de P1 y P2
                    prob_P1_dado_S1_S2_S3_0 = probabilidades[3][P1, S1 * 4
+ S2 * 2 + S3]
                    prob_P2_dado_S1_S2_S3_0 = probabilidades[4][P2, S1 * 4
+ S2 * 2 + S3]

                    # Calcular las probabilidades marginales de S1, S2 y
S3
```

```
                    prob_S1 = probabilidades[0][S1]
                    prob_S2 = probabilidades[1][S2]
                    prob_S3 = probabilidades[2][S3]

                    # Sumar las probabilidades
                    P_W1_0 += prob_W1_dado_P1_P2_0 *
prob_P1_dado_S1_S2_S3_0 * prob_P2_dado_S1_S2_S3_0 * prob_S1 * prob_S2 *
prob_S3
                    P_W1_1 += prob_W1_dado_P1_P2_1 *
prob_P1_dado_S1_S2_S3_0 * prob_P2_dado_S1_S2_S3_0 * prob_S1 * prob_S2 *
prob_S3

# Mostrar los resultados
print("P(W1 = 0):", P_W1_0)
print("P(W1 = 1):", P_W1_1)


# Inicializar las probabilidades marginales para W2
P_W2_0 = 0  # Para W2 = 0
P_W2_1 = 0  # Para W2 = 1

# Iterar sobre todas las combinaciones posibles de S1, S2, S3, P1 y P2
for S1 in range(2):
    for S2 in range(2):
        for S3 in range(2):
            for P1 in range(2):
                for P2 in range(2):
                    # Calcular las probabilidades condicionales para W2
                    prob_W2_dado_P1_P2_0 = probabilidades[6][0, P1 * 2 +
P2]
                    prob_W2_dado_P1_P2_1 = probabilidades[6][1, P1 * 2 +
P2]

                    # Calcular las probabilidades marginales de P1 y P2
                    prob_P1_dado_S1_S2_S3_0 = probabilidades[3][P1, S1 * 4
+ S2 * 2 + S3]
                    prob_P2_dado_S1_S2_S3_0 = probabilidades[4][P2, S1 * 4
+ S2 * 2 + S3]

                    # Calcular las probabilidades marginales de S1, S2 y
S3
                    prob_S1 = probabilidades[0][S1]
                    prob_S2 = probabilidades[1][S2]
                    prob_S3 = probabilidades[2][S3]
```

```
                    # Sumar las probabilidades
                    P_W2_0 += prob_W2_dado_P1_P2_0 *
prob_P1_dado_S1_S2_S3_0 * prob_P2_dado_S1_S2_S3_0 * prob_S1 * prob_S2 *
prob_S3
                    P_W2_1 += prob_W2_dado_P1_P2_1 *
prob_P1_dado_S1_S2_S3_0 * prob_P2_dado_S1_S2_S3_0 * prob_S1 * prob_S2 *
prob_S3

# Mostrar los resultados
print("P(W2 = 0):", P_W2_0)
print("P(W2 = 1):", P_W2_1)


# Inicializar las probabilidades marginales para R1
P_R1_0 = 0  # Para R1 = 0
P_R1_1 = 0  # Para R1 = 1

# Iterar sobre todas las combinaciones posibles de S1, S2, S3, P1, P2 y W1
for S1 in range(2):
    for S2 in range(2):
        for S3 in range(2):
            for P1 in range(2):
                for P2 in range(2):
                    for W1 in range(2):
                        # Calcular las probabilidades condicionales para
R1
                        prob_R1_dado_W1_0 = probabilidades[7][0, W1]
                        prob_R1_dado_W1_1 = probabilidades[7][1, W1]

                        # Calcular las probabilidades marginales de W1
                        prob_W1_dado_P1_P2 = probabilidades[5][W1, P1 * 2
+ P2]

                        # Calcular las probabilidades marginales de P1 y
P2
                        prob_P1_dado_S1_S2_S3_0 = probabilidades[3][P1, S1
* 4 + S2 * 2 + S3]
                        prob_P2_dado_S1_S2_S3_0 = probabilidades[4][P2, S1
* 4 + S2 * 2 + S3]

                        # Calcular las probabilidades marginales de S1, S2
y S3
                        prob_S1 = probabilidades[0][S1]
                        prob_S2 = probabilidades[1][S2]
                        prob_S3 = probabilidades[2][S3]
```

```
                        # Sumar las probabilidades
                        P_R1_0 += prob_R1_dado_W1_0 * prob_W1_dado_P1_P2 *
prob_P1_dado_S1_S2_S3_0 * prob_P2_dado_S1_S2_S3_0 * prob_S1 * prob_S2 *
prob_S3
                        P_R1_1 += prob_R1_dado_W1_1 * prob_W1_dado_P1_P2 *
prob_P1_dado_S1_S2_S3_0 * prob_P2_dado_S1_S2_S3_0 * prob_S1 * prob_S2 *
prob_S3

# Mostrar los resultados
print("P(R1 = 0):", P_R1_0)
print("P(R1 = 1):", P_R1_1)


# Inicializar las probabilidades marginales para R2
P_R2_0 = 0  # Para R2 = 0
P_R2_1 = 0  # Para R2 = 1

# Iterar sobre todas las combinaciones posibles de S1, S2, S3, P1, P2 y W2
for S1 in range(2):
    for S2 in range(2):
        for S3 in range(2):
            for P1 in range(2):
                for P2 in range(2):
                    for W2 in range(2):
                        # Calcular las probabilidades condicionales para
R2
                        prob_R2_dado_W2_0 = probabilidades[8][0, W2]
                        prob_R2_dado_W2_1 = probabilidades[8][1, W2]

                        # Calcular las probabilidades marginales de W2
                        prob_W2_dado_P1_P2 = probabilidades[6][W2, P1 * 2
+ P2]

                        # Calcular las probabilidades marginales de P1 y
P2
                        prob_P1_dado_S1_S2_S3_0 = probabilidades[3][P1, S1
* 4 + S2 * 2 + S3]
                        prob_P2_dado_S1_S2_S3_0 = probabilidades[4][P2, S1
* 4 + S2 * 2 + S3]

                        # Calcular las probabilidades marginales de S1, S2
y S3
                        prob_S1 = probabilidades[0][S1]
                        prob_S2 = probabilidades[1][S2]
```

```
                        prob_S3 = probabilidades[2][S3]

                        # Sumar las probabilidades
                        P_R2_0 += prob_R2_dado_W2_0 * prob_W2_dado_P1_P2 *
prob_P1_dado_S1_S2_S3_0 * prob_P2_dado_S1_S2_S3_0 * prob_S1 * prob_S2 *
prob_S3
                        P_R2_1 += prob_R2_dado_W2_1 * prob_W2_dado_P1_P2 *
prob_P1_dado_S1_S2_S3_0 * prob_P2_dado_S1_S2_S3_0 * prob_S1 * prob_S2 *
prob_S3

# Mostrar los resultados
print("P(R2 = 0):", P_R2_0)
print("P(R2 = 1):", P_R2_1)


# Inicializar las probabilidades marginales para R3
P_R3_0 = 0  # Para R3 = 0
P_R3_1 = 0  # Para R3 = 1

# Iterar sobre todas las combinaciones posibles de S1, S2, S3, P1, P2 y W2
for S1 in range(2):
    for S2 in range(2):
        for S3 in range(2):
            for P1 in range(2):
                for P2 in range(2):
                    for W2 in range(2):
                        # Calcular las probabilidades condicionales para
R3
                        prob_R3_dado_W2_0 = probabilidades[9][0, W2]
                        prob_R3_dado_W2_1 = probabilidades[9][1, W2]

                        # Calcular las probabilidades marginales de W2
                        prob_W2_dado_P1_P2 = probabilidades[6][W2, P1 * 2
+ P2]

                        # Calcular las probabilidades marginales de P1 y
P2
                        prob_P1_dado_S1_S2_S3_0 = probabilidades[3][P1, S1
* 4 + S2 * 2 + S3]
                        prob_P2_dado_S1_S2_S3_0 = probabilidades[4][P2, S1
* 4 + S2 * 2 + S3]

                        # Calcular las probabilidades marginales de S1, S2
y S3
                        prob_S1 = probabilidades[0][S1]
```

```
                        prob_S2 = probabilidades[1][S2]
                        prob_S3 = probabilidades[2][S3]

                        # Sumar las probabilidades
                        P_R3_0 += prob_R3_dado_W2_0 * prob_W2_dado_P1_P2 *
prob_P1_dado_S1_S2_S3_0 * prob_P2_dado_S1_S2_S3_0 * prob_S1 * prob_S2 *
prob_S3
                        P_R3_1 += prob_R3_dado_W2_1 * prob_W2_dado_P1_P2 *
prob_P1_dado_S1_S2_S3_0 * prob_P2_dado_S1_S2_S3_0 * prob_S1 * prob_S2 *
prob_S3

# Mostrar los resultados
print("P(R3 = 0):", P_R3_0)
print("P(R3 = 1):", P_R3_1)
```